Міністерство освіти та науки України

Національний університет водного господарства та природокористування

Кафедра автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій

04-03-288

Методичні вказівки

до виконання лабораторних робіт з дисципліни «**Технології об`єктно-орієнтованого та web-програмування. Частина 2**» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Автоматизація та комп'ютерно-інтегровані технології» спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» денної та заочної форм навчання

Рекомендовано науково-методичною радою з якості ННІ АКОТ
Протокол № 8 від 29.04.2020 р.

Рівне – 2020

Методичні вказівки до виконання лабораторних робіт з дисципліни «Технології об`єктно-орієнтованого та web-програмування. Частина 2» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Автоматизація та комп'ютерно-інтегровані технологій» спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» [Електронне видання] / Парфенюк О. І., Присяжнюк О. В., Сафоник А. П. – Рівне : НУВГП, 2020. – 94 с.

Укладачі: Парфенюк О. І., ст. викладач кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій; Присяжнюк О. В., ст. викладач кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій; Сафоник А. П. д.т.н., професор кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій.

Відповідальний за випуск: Древецький В. В., д.т.н., професор, завідувач кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій.

Керівник групи забезпечення спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»: Древецький В. В., д.т.н., професор завідувач кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій.

Зміст

1. Лабораторна робота №7. Веб-фреймворк Django. Налаштування середовища розробки..……4

2. Лабораторна робота №8. Створення Django-аплікацій. Робота з базою даних та інтерфейсом адміністратора……....18

3. Лабораторна робота №9. Створення моделей та робота з ORM ……..…36

4. Лабораторна робота №10. Розробка серверної частини персонального блогу. Модульне тестування веб-додатку ….....52

5. Лабораторна робота №11. Розробка клієнтської частини веб-застосування. Робота зі статичними файлами……..….79

Лабораторна робота №7
Веб-фреймворк Django. Налаштування середовища розробки

7.1. Мета роботи

Познайомитись з веб-фреймворком Django і отримати навики налаштування середовища розробки Django-проекту.

7.2. Теоретичні відомості

Django

Django — Python-фреймворк з відкритим кодом для швидкої розробки веб-систем. Спочатку технологію розробляли як засіб для керування сайтами новин LJWorld.com, lawrence.com та Kusports.com компанії The World Company і це значно вплинуло на її архітектуру, оскільки реалізовано цілий ряд функціональних можливостей, які допомагають у швидкій розробці веб-сайтів інформаційного характеру.

Щоб мати уявлення про те, що ви можете зробити з Django, добре було б дізнатися, хто його використовує. Серед найбільших компаній, що використовують Django, ми маємо: Instagram, Disqus, Mozilla, Bitbucket, Last.fm, National Geographic.

Сайт з використанням Django будується з однієї або декількох частин, які рекомендовано робити модульними (аплікації, app).

Архітектура схожа на «Модель-Вид-Контролер» (MVC). Однак, тут роль «контролера» класичної моделі MVC виконує «вид» (view), а «видом» називається «шаблон» (template). Таким чином, MVC розробники Django називають MTV («МодельШаблон-Вид»). Однією з основних переваг для розробника є відсутність потреби створювати контролери та сторінки для адміністративної частини сайту, в збірці є вбудований модуль для керування вмістом, який можна долучити до будь-якого сайту, написаний на Django, і який може керувати відразу декількома сайтами на одному сервері.

Ще однією значною перевагою Django є адміністративний модуль, який дає змогу створювати,

змінювати і вилучати будь-які об’єкти наповнення сайту, фіксуючи всі дії та надає інтерфейс для керування обліковими записами користувачів і групами (з призначенням прав). У збірку також внесені засоби для системи коментарів і «статичні сторінки», які можна використовувати без необхідності писати додаткові контролери та відображення.

До базових функцій Django належать: Об’єктно-реляційне відображення (ORM), яке допомагає суттєво спростити роботу з базою даних. Об’єкти БД в термінології Django іменуються «моделями». Розробнику не потрібно писати SQL-запити (але така можливість є), бо під час виконання синхронізації проекту з БД автоматично будуть створені усі таблиці з полями, які відповідають властивостям (properties) описаних моделей.

Зручним також є синтаксис url-адрес, який побудований на регулярних виразах. Розробник не обмежений у використанні певної схеми посилань. Посилання можуть групуватися за кожним модулем проекту в окремий файл. Крім того, можна використовувати багато інших способів групування url-адрес, як стосовно конкретного модуля, так і стосовно усього проекту.

Зручна система шаблонів, яка передбачає наявність окремої мови для їх опису. Вона є достатньо простою, містить оператори циклу, умови, засоби форматування даних. Мова шаблонів виконує функцію відображення даних.

Гнучка підсистема кешування дає змогу дуже швидко налаштувати Django-проект для роботи з Memcached чи будь-якою іншою надбудовою. Інструменти Django дають змогу кешувати SQL-вибірки, шаблони та їх частини і просто окремі змінні.

Простою є інтернаціоналізація, що базується на концепції «лінивого» перекладу. Це означає, що якщо певний рядок тексту не має перекладу, то буде використано базовий текст і не буде показано повідомлення про помилку. Також можна використовувати спеціальні функції для контролю перекладу рядкових даних.

У збірці Django є власний веб-сервер для розробки і налагоджування. Він автоматично відслідковує зміни у файлах

програмного коду і перезапускається, що дуже зручно при розробці проекту.

Для розробки різного рівня веб-застосунків можна використовувати вже готові модулі чи надбудови. На сайті djangopackages.com дуже легко відшукати пакет, який найкраще підійде для вирішення конкретної задачі. Всі ці застосунки розповсюджуються вільно і кожен охочий може їх завантажити з репозиторію чи приєднатися до команди розробників.

Для роботи з Python існує багато спеціальних середовищ чи надбудов, та все ж найпопулярнішим вважається PyCharm. Одночасно розвиваються дві окремі гілки цього проекту: комерційна (розробляється компанією JetBreins) і відкрита (розвивається спільнотою).

Система керування версіями

Система керування версіями (англ. source code management, SCM) — програмний інструмент для керування версіями одиниці інформації: вихідного коду програми, скрипту, веб-сторінки, веб-сайту, 3D моделі, текстового документу тощо.

Система керування версіями — це потужний інструмент, який дозволяє одночасно, без завад один одному, проводити роботу над груповими проектами.

Системи керування версіями зазвичай використовуються при розробці програмного забезпечення для відстеження, документування та контролю над поступовими змінами в електронних документах: у сирцевого коду застосунків, кресленнях, електронних моделях та інших документах, над змінами яких одночасно працюють декілька людей.

Кожна версія позначається унікальною цифрою чи літерою, зміни документу занотовуються. Зазвичай також зберігається автор зробленої зміни та її час.

Система керування версіями існують двох основних типів: з централізованим сховищем та розподіленим.

Централізована система контролю версії (клієнт-серверна) — система, дані в якій зберігаються в єдиному «серверному» сховищі. Весь обмін файлами відбувається з використанням

центрального сервера. Є можливість створення та роботи з локальними репозиторіями (робочими копіями).

Розподілена система контролю версії (англ. Distributed Version Control System, DVCS) — система, яка використовує замість моделі клієнт-сервер, розподілену модель зберігання файлів. Така система не потребує сервера, адже всі файли знаходяться на кожному з комп'ютерів.

До таких систем відносять Git, Mercurial, SVK, Monotone, Codeville, BitKeeper.

Система контролю дозволяє зберігати попередні версії файлів та завантажувати їх за потребою. Вона зберігає повну інформацію про версію кожного з файлів, а також повну структуру проекту на всіх стадіях розробки. Місце зберігання даних файлів називають репозиторієм. В середині кожного з репозиторіїв можуть бути створені паралельні лінії розробки — гілки.

Гілки зазвичай використовують для зберігання експериментальних, незавершених(alpha, beta) та повністю робочих версій проекту(final). Більшість систем контролю версії дозволяють кожному з об'єктів присвоювати теги, за допомогою яких можна формувати нові гілки та репозиторії.

Використання системи контролю версії є необхідним для роботи над великими проектами, над якими одночасно працює велика кількість розробників. Системи контролю версії надають ряд додаткових можливостей:

- Можливість створення різних варіантів одного документу;
- Документування всіх змін (коли ким було змінено/додано, хто який рядок змінив);
- Реалізує функцію контролю доступу користувачів до файлів. Є можливість його обмеження;
- Дозволяє створювати документацію проекту з поетапним записом змін в залежності від версії;
- Дозволяє давати пояснення до змін та документувати їх.

7.3. Програма роботи

7.3.1. Ознайомитися з призначенням та принципами роботи в інтегрованих середовищах розробки програмного забезпечення PyCharm Community.

7.3.2. Налаштувати середовище розробки Django-проекту.

7.3.3. Створити Django-проекту та Django-аплікацію.

7.4. Обладнання та програмне забезпечення

7.4.1. Персональний комп'ютер.

7.4.2. Інтерпретатор Python встановлений на ПК

7.4.3. Web-фреймворк Django.

7.5. Порядок виконання роботи і опрацювання результатів

Інсталювати Django ми не будемо напряму в Python, а у окремо створене так зване віртуальне середовище (пакет virtualenv).

Віртуальне середовище - це можливість мати кілька Python проектів і працювати над ними паралельно. При цьому один іншому не буде перешкоджати. Це окремі папки із своєю копією Python та усіх заінстальованих додаткових бібліотек.

Virtualenv – це інструмент, який дозволяє налаштовувати безліч "копій" вашого заінстальованого Python, і працювати над кількома Python проектами одночасно не заважаючи один одному. Цей підхід дозволяє уникати конфліктів між заінстальованими пакетами та їхніми версіями. Наприклад, якщо вам потрібно в одному проекті використовувати Django 1.10, а в іншому 2.1.

Іншими словами можете уявити, що кожне віртуальне середовище це окрема папочка зі своїм набором налаштувань, які не заважають іншому такому ж віртуальному середовищу. Це як поставити на Windows програму лише для одного користувача, а інші користувачі мають свої власні набори програм.

Маючи Pip можемо з легкістю однією командою

встановити virtualenv пакет. В PowerShell або cmd в контексті будь-якої папки запускаєте наступну команду (з правами адміністратора):

pip install virtualenv

Також тепер ви можете ввести команду virtualenv і вона виведе вам інструкцію по використанню.

У віртуальному середовищі маємо наступні важливі для нас папки:

• bin: бінарники (python, pip, activate, тут також будуть скоро django скрипти);

• lib: пітонівські пакети, сюди будуть також за інстальовані пакети (підпапка python3.8/site-packages – в даному випадку версія Python, але може відрізнятись), сюди ми не раз будемо заглядати під час розробки.

Створюємо віртуальне середовище під свій проект

Ось ми і підійшли безпосередньо до підготовки до робочого середовища нашого Django проекту, а саме створення віртуального середовища python для нашого конкретного Django проекту.

Оберіть і створіть у себе на диску папку, в якій міститимете усі майбутні проекти (наприклад “D:\work”), а в ній директорію під перший проект (myblog) Перейшовши в обрану директорію, створюємо там нове віртуальне середовище для проекту:

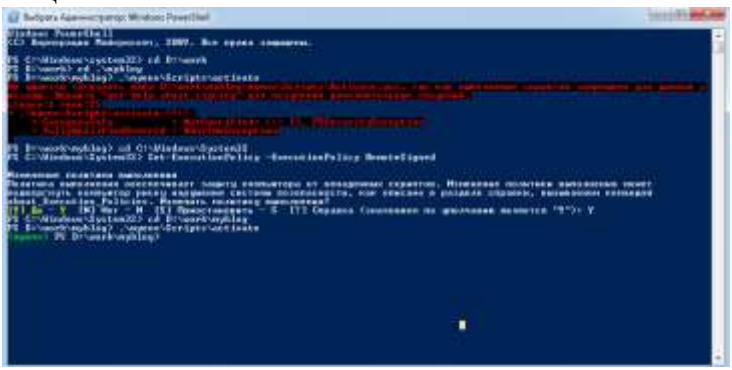
Linux:
Virtualenv myvenv --no-site-packages
source myvenv/bin/activate

Windows:
D:\work\myblog > python -m venv myvenv

Після цього в папці myblog з’явиться відповідна папка myvenv. Для подальшої роботи всередині віртуального оточення його потрібно активувати командою

myvenv\Scripts\activate

Якщо виникла помилка:



Потрібно, зайшовши під правами адміністратора, виконати команду:

C:\WINDOWS\system32>
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned

Тепер можна активувати віртуальне оточення. При цьому на початку рядка з’явиться назва віртуального оточення в дужках. Усі наступні установки будуть впливати лише на конфігурацію всередині віртуального оточення. Всі команди потрібно запускати **лише** у віртуальному оточенні. Вимкнути його можна командою

deactivate

Установіть django командою

pip install django

І створіть проект (назва проекту *mysite*, може бути ваша власна) командою

django-admin startproject mysite

Тепер в каталозі myblog поруч з myvenv зявилась папка mysite. Вона міститиме всі вихідні файли майбутнього сайту.

Команда django-admin (а також скрипт manage.py) мають цілий набір підготованих команд, які дозволяють працювати із базою даних, запускати міграції, запускати розробницький Django сервер і ще дуже багато інших речей.

Django Проект - це заготований для нас набір папок та файлів (а в поняттях Python - пакетів та модулів), які складають кістяк веб-аплікації в Django.

Проект дає нам хороший старт для написання нашого власного коду і наперед задає набір правил що і куди має йти.

Після створення нашого середовища матимемо наступну структуру файлів і папок:

• manage.py - аналог django-admin, але в контексті проекту;

• mysite - папка Django проекту;

– settings.py - модуль із налаштуваннями Django проекту;

– urls.py - налаштування URL диспетчера;

– wsgi.py - модуль, що робить нашу аплікацією WSGI аплікацією; службовий модуль, який виступає «посередником» між веб-сервером і проектом.

Перед запуском сайту нам ще потрібно початково налаштувати базу даних.

Робиться це доволі просто командою скрипта manage.py:

- створення початкової схеми бази даних:

python mysite/manage.py migrate

- додавання суперкористувача, username і password будуть

використовуватись для входу в адмінку, можна створювати довільну кількість суперкористувачів:

python mysite/manage.py createsuperuser

```
(myvenv) PS D:\work\myblog> python mysite/manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
(myvenv) PS D:\work\myblog> python mysite/manage.py createsuperuser
Username (leave blank to use 'igor'): admin
Email address: admin@admin.ua
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
(myvenv) PS D:\work\myblog>
```

Для перевірки правильності установки Django необхідно запустити сервер розробки, щоб подивитися на створений додаток в дії.

Сервер розробки Django (також званий «runserver», по імені команди, яка його запускає) - це вбудований легкий веб-сервер, який ви можете використовувати в процесі розробки вашого сайту. Він включений в Django для того, щоб ви могли швидко приступити до розробки вашого сайту без витрачання часу на конфігурацію вашого бойового веб-сервера (тобто, Apache) завчасно. Цей сервер розробки відстежує зміни в вашому коді і автоматично перезавантажує його, допомагаючи бачити вносяться вами зміни без перезавантаження веб-сервера. Можна запускати проект

python mysite/manage.py runserver

Набравши в браузері http://127.0.0.1:8000/ , побачите

Щоб вийти із даного серверного процесу, потрібно скористатись набором
клавіш Ctrl-C.

Всі вимоги в одному місці. Замороження пакетів

Зазвичай для запуску проекту потрібно кілька зовнішніх пакетів. Щоб кожен раз не збирати їх, список цих пакетів зберігається разом з вихідним кодом у файлі **requirements.txt** в корені проекту. Формат цього файлу простий: по одному пакету на рядок.

У одного пакета зазвичай багато версій. Коли ми “просимо” pip встановити пакет, він встановлює найсвіжішу з доступних.

Це може привести до проблем: скажімо, проект розроблявся на версії 1.2. Через півроку знадобилося розгорнути його заново, pip встановив останню версію - 1.5. Ця версія може бути не сумісна зі старою, тоді код зламається. Щоб трохи спростити цю задачу розробники використовують pip і файл requirements.txt. У цьому файлі записуються всі необхідні для роботи бібліотеки і, що найголовніше, вказують версії цих бібліотек.

Маючи такий файл налаштування оточення для старту проекту може складатися з однієї команди:

pip install -r requirements.txt

Отримати список пакетів, що заінстальовані на даний момент, можна командою (слідкуйте чи активоване віртуальне оточення, тобто на початку рядка в круглих дужках має бути назва віртуального оточення):

pip freeze

або

pip freeze > requirements.txt

```
(myvenv) PS D:\work\myblog> pip freeze
asgiref==3.2.3
Django==3.0.3
pytz==2019.3
sqlparse==0.3.0
(myvenv) PS D:\work\myblog> pip freeze > mysite/requirements.txt
(myvenv) PS D:\work\myblog>
```

Перша команда виводить список пакетів у консоль, друга перезаписує файл requirements.txt в директорії mysite.

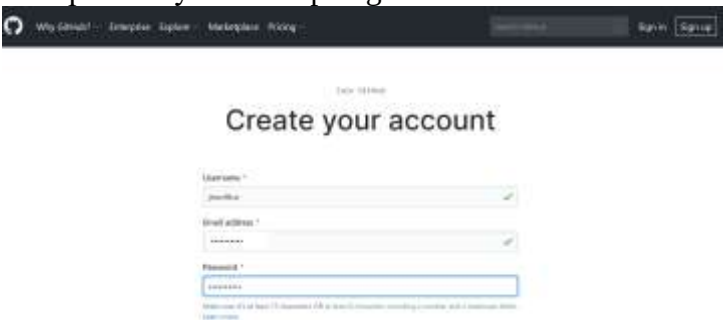
Завдання:

Відкрийте файл settings.py. Уважно ознайомтесь з існуючими там змінними і переведіть опис кожної з них.

Конфігуруємо Git

Важливо вміти користуватись Git з консолі особливо, коли вам приходиться працювати на віддаленому сервері, де немає графічної оболонки.

Створіть акаунт на https://github.com/



За лінком http://git-scm.com/download/win завантажте відповідний вашій ОС Git інсталяційний файл *.exe. Після завантаження запускаєте і проходите стандартний процес встановлення софта на Windows (вибираєте запропоновані налаштування). Після інсталяції в командному рядку повинна бути доступна команда “git”, а також Git Bash.

Після інсталяцій Git у будь-якій із операційних систем потрібно зробити наступний мінімум налаштувань з вашого командного рядка:

Встановлюємо глобально ваш email та ім’я для Git

git config --global user.name User
git config --global user.email User@mail.com

Таким чином, ваші коміти в репозиторій будуть позначені вашим іменем. Звісно, замінюєте емейл **User@mail.com** та ім'я **User** вашими особистими даними.

На завершення маємо закомітити щойно створений проект у репозиторій коду. У репозиторій покладемо корінь нашого Django проекту. Переконаємось, що ми в корені нашого проекту (там де лежить файл manage.py та ініціалізуємо новий репозиторій командою

git init

Створюємо файл для «ігнору» файлів, які не повинні бути додані в репозиторій на сервері. Такий файл .gitignore має розташовуватись в корені проекту, там же де ініціалізувався репозиторій командою git init, він не повинен мати розширення, а також є прихованим (крапка перед ім'ям файлу).

Наприклад щоб не заливати на сервер бінарні файли (тобто всі із розширенням .pyc), необхідно відкрити створений файл .gitignore і написати в ньому наступний рядок:

*.pyc

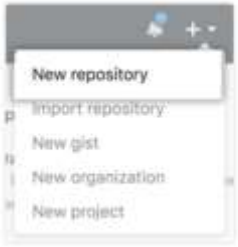
Додаємо усі файли до індекса командою

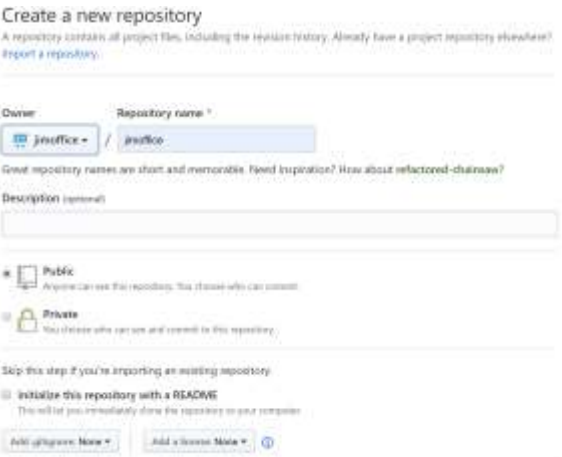
git add *

Комітимо усі файли в локальний репозиторій, не забудьте додати змістовний коментар до вашого коміту, наприклад *initial project template*:

git commit -m "initial project template"

Створіть новий відкритий (public) репозиторій на Github:





Після цього вгорі створеного репозиторію скопіюйте URL

В консолі виконайте команду
git remote add origin repo_URL

```
(myvenv) D:\work\myblog\mysite>git remote add origin https://github.com/jimoffice/jimoffice.git
```

Після цього зміни на сервер заливаються командою:
git push origin master

При цьому потрібно буде ввести ваш логін та пароль GitHub.

Перейшовши в репозиторій в GitHub побачите, що там з’явились файли вашого проекту.

Тепер ваш проект локально зв’язаний з проектом на сервері. Всі зміни, які будете вносити надалі, необхідно комітити та заливати на сервер **після кожної лабораторної роботи** за наступним алгоритмом:

1. перевіряємо внесені зміни **git status**
2. додаємо файли, якщо git status показує Untracked files
   git add назва_файлу
3. комітимо зміни з повідомленням про те, що було модифіковано
   git commit -m "#182: Fix benchmarks weights"
4. заливаємо ві закомічені зміни на серер
   git push origin

Посилання на свій репозиторій на GitHub необхідно додати в звіт.

7.6. Контрольні запитання.

7.6.1. Що таке Django?
7.6.2. В якому файлі зберігаються налаштування проекту?
7.6.3. Назвіть основні переваги Django-фреймворка
7.6.4. Для чого використовується команда pip freeze?
7.6.5. Що таке віртуальне оточення?

Література:

1. Лутц М. Программирование на Python, том I, 4-е издание.– Пер. сангл. – СПб.: Символ-Плюс, 2011.– 992с.
2. Документація Django [Електронний ресурс] / Режим доступу : https://djbook.ru/rel3.0
3. Основи Git [Електронний ресурс] / Режим доступу : https://git-scm.com/book/uk/

Лабораторна робота №8
Створення Django-аплікацій. Робота з базою даних та інтерфейсом адміністратора

8.1. Мета роботи

Ознайомитись з процесом створення Django-аплікації та процесом налаштування проекту, створити базу даних для сайту.

8.2. Теоретичні відомості

Деякі можливості Django:

- ORM, API доступу до БД з підтримкою транзакцій;
- вбудований інтерфейс адміністратора, з уже наявними перекладами багатьма мовами;
- URL-диспетчер на основі регулярних виразів;
- розширювана система шаблонів з тегами і спадкуванням;
- система кешування;
- підключається архітектура додатків, які можна встановлювати на будь-які Django-сайти;
- авторизація та автентифікація, підключення зовнішніх модулів автентифікації: LDAP, OpenID та інші;
- система фільтрів («проміжного шару») для побудови додаткових обробників запитів, як наприклад включені в дистрибутив фільтри для кешування, стиснення, URL нормалізації і підтримки анонімних сесій;
- бібліотека для роботи з формами (успадкування, побудова форм по існуючої моделі БД);
- вбудована автоматична документація по тегам шаблонів і моделей даних.

Django Аплікація – це невелика бібліотека коду, яка представляє один аспект цілого вашого проекту. Зазвичай Django Проект складається із кількох (деколи дуже багатьох) Django Аплікацій. Деякі із цих аплікацій є внутрішніми для вашого конкретного проекту і ніколи не використовуватимуться

у інших проектах, в той час як інші є зовнішніми і можуть використовуватися вами у інших Django проектах.

Сторонні Django Пакети (Додатки, Аплікації) – python пакети та Django аплікації створені іншими розробниками, які ви використовуєте у ваших власних проектах.

Кожна Django аплікація не повинна виконувати більше, ніж одну задачу у проекті. Тобто заведено тримати аплікації малими.

Важливо відзначити, що Django-додатки слідують парадигмі *Модель - Представлення - Шаблон*. У двох словах, додаток отримує дані від моделі, представлення робить щось з даними, та створюється шаблон, що містить оброблену інформацію. Таким чином, шаблони Django відповідають представленню у традиційному MVC і представлення Django можуть бути прирівнені до контролерів в традиційному MVC.

Додаток Python входить до складу проекту і реалізує функціональність одного з розділів сайту і всіх його підрозділів. Кількість додатків в проекті не обмежена. Фізично додаток являє собою пакет, папка якого знаходиться в папці проекту. Ім'я цього пакета стане ім'ям програми, а сам пакет називається пакетом додатку. Пакет додатку формується самою Django при створенні програми.

Спочатку він містить наступні модулі:

- **migrations/**: тут Django зберігає деякі файли для відстеження змін, створених у файлі **models.py**, щоб підтримувати синхронізацію бази даних та моделей **models.py**.
- **admin.py**: файл конфігурації для вбудованого застосунку Django під назвою **Django Admin**.
- **apps.py**: файл конфігурації самого застосунку.
- **models.py**: тут ми визначаємо об'єкти нашого веб-застосунку. Django автоматично переводить моделі в таблиці бази даних.
- **tests.py**: файл використовується для написання модульних тестів для застосунку.

• **views.py**: файл, в якому ми обробляємо цикл запитів/відповідей нашого веб-застосунку.

Зрозуміло, ми можемо, якщо виникне така необхідність, створити в пакеті додатки інші модулі, що зберігають код моделей, контролерів або додаткових функцій і класів, що ми задіємо в коді сайту. Django в цьому плані ніяк нас не обмежує. Ось тільки всі шаблони, які застосовуються в сайті, треба створити вручну. Навіть якщо додаток входить до складу проекту, це ще не говорить про те, що він буде задіяний в сайті. Щоб додаток успішно працював, слід, по15 перше, виконати його прив'язку до інтернет-адреси, а по-друге, вказати його в списку активних додатків, що знаходиться в модулі settings пакета проекту. Тільки після цього додаток стане активним. Потрібно сказати, що частина допоміжних модулів Django реалізована також у вигляді додатків (вбудовані додатки). Таким вбудованим додатком є, зокрема, підсистема, що реалізує розмежування доступу.

Адміністративний сайт, за допомогою якого ми можемо працювати зі збереженими в базі даними, також є додатком подібного роду.

Вбудовані додатки або також прив'язуються до інтернет-адреси та, тим самим, формують новий розділ сайту, або працюють постійно, забезпечуючи допоміжну функціональність. У будь-якому випадку їх також потрібно вказати в списку активних додатків, інакше вони не будуть задіяні. Прив'язка інтернет-адрес. Ми знаємо, що кожен додаток сайту, запускається у відповідь на звернення до певної інтернет-адреси, до якої він був прив'язаний. Єдиний виняток тут – вбудовані додатки, з якими ми тільки що познайомилися і які працюють постійно і не вимагають такої прив'язки.

У бібліотеці Django прив'язка інтернет-адреси до додатка виконується в модулі urls пакета проекту. Або, кажучи іншими словами, прив'язка адрес до додатків виконується на рівні проекту. Але в реальності один додаток може виконувати відразу кілька дій. Скажімо, додаток списку товарів може виводити як і сам цей список, так і відомості про обраний товар,

а додаток гостьової книги – як виводити гостьову книгу, так і додавати в неї новий запис. Як це реалізувати? Дуже просто. Окремим контролерам додатків ставляться у відповідність віртуальні папки згаданих раніше папок, що формують підрозділи даних розділів сайту. При прив'язці інтернет-адреси до контролера ми можемо вказати, що останній повинен приймати будь-які дані. Ці дані будуть передані в складі інтернет-адреси із застосуванням методу GET. Наприклад, оскільки ми збираємося реалізувати висновок відомостей про обраний товар, нам доведеться передавати відповідним контролеру ідентифікатор цього товару.

База даних – впорядкований набір логічно взаємопов'язаних даних, що використовуються спільно та призначені для задоволення інформаційних потреб користувачів.

Реляційна база даних - це набір таблиць та зв'язків між цими таблицями. Таблиця, в свою чергу, складається із стовпців та рядків. *Стовпці* (поля) - це структура таблиці. *Рядки* - це дані таблиці.

Один рядок в таблиці представляє одну одиницю даних і містить значення (або порожнє значення) для кожного із полів таблиці. Також рядок ще називають "записом" таблиці (англ. record). Кожне поле (англ. field) таблиці має тип подібно до того, як мова програмування Python має набір своїх типів даних.

Типи даних поділяються на три категорії: числові, текстові і типи дати та часу.

Найчастіше використовуваними типами в проекті будуть:

- VARCHAR;
- TEXT;
- ENUM;
- INT;
- DATE;
- DATETIME;

Таким чином поля таблиці визначають її структуру. А записи (рядки) таблиці містять самі дані.

8.3. Програма роботи

8.3.1. Налаштувати інтегроване середовище для розробки.

8.3.2. Створити Django app та ознайомитись із принципами архітектури, роботою з представленнями.

8.3.3. Налаштувати базу даних, ознайомитись з інтерфейсом вбудованої панелі адміністратора.

8.3.4. Створити першу сторінку сайту.

8.4. Обладнання та програмне забезпечення

8.4.1. Персональний комп'ютер.

8.4.2. Інтерпретатор Python встановлений на ПК

8.4.3. Середовище розробки програмного забезпечення PyCharm Community Edition (чи інше IDE, наприклад Eclipse, Sublime Text, Geany).

8.4.4. Web-фреймворк Django.

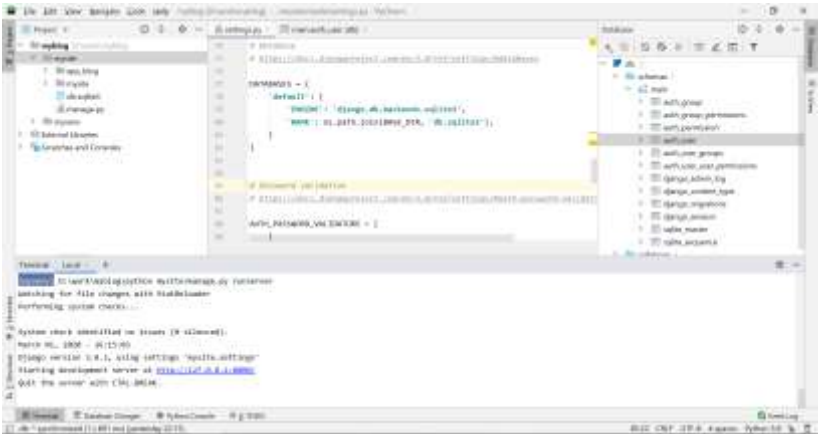
8.5. Порядок виконання роботи і опрацювання результатів

Налаштування IDE

Для прискорення процесу написання програм використовують зручно використовувати інтегроване середовище розробки, яке включає в себе різні інструменти для роботи з кодом: засіб для написання коду (текстовий редактор), інтерактивний інтерпретатор, відлагоджувач тощо.

Веб-розробку на Django підтримує інтегроване середовище розробки для мови програмування Python PyCharm. Присутня безкоштовна версія Community. PyCharm працює під операційними системами Windows, MacOS і Linux. Ось так виглядає проект myblog, відкритий в PyCharm.

Зліва присутня панель навігації по проекту, справа власне відкритий для редагування один із файлів проекту. Зручною можливістю є робота в терміналі (в нижньому вікні клікнувши на вкладці Terminal). Детальніше про налаштування PyCharm https://py-charm.blogspot.com/2017/09/blog-post.htm.



Якщо ви встановили PyCharm та відкрили в ньому проект, активувати віртуальне (команда *myvenv\Scripts\activate*) оточення можна тепер не за допомогою командного рядка Windows, а за допомогою терміналу PyCharm (слідкуйте, щоб при роботі з проектом віртуальне оточення було завжди активоване). Тут же вводимо команду запуску сервера (*python mysite/manage.py runserver*) і переконуємось, що проект працює перейшовши в браузері за адресою http://127.0.0.1:8000/. Ще однією перевагою є зручність в роботі з базою даних.

PyCharm дозволяє працювати з Git не з командного рядка, а засобами IDE (вкладка VCS). При додаванні нових файлів IDE запитуватиме, чи потрібно додавати файли до проекту. Файли, в яких внесені зміни та які не додані до репозиторію, виділяються іншим кольором для зручного керування проектом.

Після виконання лабораторної роботи, щоб закомітити внесені в проект зміни, натискаємо Commit та уважно переглядаємо змінені файли, впевнюємось що всі нові файли додано, пишемо коментар про те, що було змінено. Після цього натискаємо Push.

Створення власного додатку

Для створення Django-аплікації. Зробіть перехід у першу папку mysite (в консолі за допомогою команди cd) слідкуючи при цьому, щб віртуальне (myvenv) оточення було активоване

```
(myvenv) D:\work\myblog>cd mysite

(myvenv) D:\work\myblog\mysite>
```

і введіть:

python manage.py startapp app_blog

Виконання цієї команди створить додаток під назвою app_blog.

Щоб Django впізнав наш новий додаток, нам потрібно додати його назву в список Installed Apps в файлі settings.py.

```
# mysite/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app_blog'
]
```

Додавання URL та Шаблонів

Коли ми запустили сервер, то побачили дефолтну сторінку Django. Нам потрібно, щоб Django отримувала доступ до нашого додатку app_blog, коли хтось відвідує URL головної сторінки /. Для цього нам потрібно визначити URL, який повідомить Django, де шукати шаблон головної сторінки.

Відкрийте файл urls.py у внутрішній папці mysite. Це має виглядати наступним чином.

```
 """mysite URL Configuration

The `urlpatterns` list routes URLs to views. For more
information please see:
    https://docs.djangoproject.com/en/3.0/topics/http/urls/
```

```
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path(‘‘, views.home,
name=‘home’)
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path(‘‘, Home.as_view(),
name=‘home’)
Including another URLconf
    1. Import the include() function: from django.urls
import include, path
    2. Add a URL to urlpatterns:  path(‘blog/’,
include(‘blog.urls’))
"""
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path(‘admin/’, admin.site.urls
]
```

Як ви бачите, є існуючий патерн URL для адмін-сайту Django, який створено по дефолту з Django. Додамо наш власний URL, щоб вказати на наш app_blog додаток. Відредагуйте цей файл наступним чином.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path(‘admin/’, admin.site.urls),
    path(r’’, include(‘app_blog.urls’)),
]
```

Зверніть увагу, що ми додали імпорт для include з django.conf.urls та додали патерн URL для порожнього маршруту. Коли хтось заходить на головну сторінку, (в нашому випадку http://localhost:8000), Django буде шукати більше URL в додатку app_blog. Так як там немає жодного, запустивши додаток ми отримаємо величезне трасування стеку через ImportError.

ImportError: No module named ‘ app_blog .urls’

Для того, щоб виправити це, перейдіть в папку app_blog і створіть файл під назвою urls.py. Папка app_blog тепер має виглядати наступним чином.

```
├── app_blog
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   ├── __init__.py
│   ├── models.py
│   ├── tests.py
│   ├── urls.py
│   └── views.py
```

Всередині нового файлу urls.py, напишіть наступне.

```
# app_blog/urls.py
from django.urls import path
from app_blog import views

urlpatterns = [
    path(r’’, views.HomePageView.as_view()),
]
```

Цей код імпортує представлення з нашого app_blog додатку та очікуватиме визначення представлення, яке називається HomePageView. Так як у нас немає ні одного, відкрийте файл views.py в app_blog і введіть цей код.

```
# app_blog /views.py
from django.shortcuts import render
from django.views.generic import TemplateView

# Створіть свої представлення тут.
class HomePageView(TemplateView):
    def get(self, request, **kwargs):
        return render(request, ‘index.html’, context=None)
```

Цей файл визначає представлення під назвою HomePageView. Представлення Django приймають request і повертають response. У нашому випадку метод get очікує запит HTTP GET до URL, визначеного в нашому файлі urls.py.

Після того, як запит HTTP GET був отриманий, метод надає шаблон під назвою index.html, який представляє собою звичайний HTML-файл, який може мати спеціальні теги Django шаблонів, написані зі звичайними HTML-тегами. Якщо запустити сервер зараз, ви побачите наступну error сторінку:

Це відбувається тому, що у нас взагалі немає шаблонів. Django шукає шаблони в папці templates всередині вашого додатку, тому створіть її у вашій папці app_blog.

Перейдіть до папки для шаблонів, яку ви створили та створіть файл під назвою index.html

В index.html вставте цей код:

```
<!DOCTYPE html>
<html lang="uk">

<head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <title>Blog</title>
</head>

<body>
   Home page
</body>

</html>
```

Тепер запустіть ваш сервер.

python manage.py runserver

Ви побачите ваш шаблон.

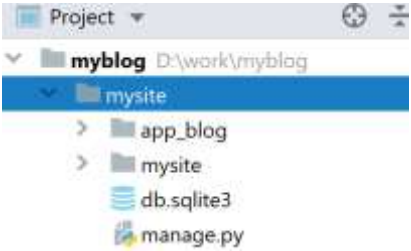
Налаштування бази даних

Відкрийте файл mysite / settings.py. Це звичайний модуль Python з набором змінних, які представляють настройки Django та знайдіть рядки

```
# Database

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

За замовчуванням в налаштуваннях зазначено використання бази даних SQLite, це найпростіший вибір. SQLite вже включений в Python, і не потрібно додатково щось встановлювати, проте для реальних робочих проектів ця база даних не підходить. Зазвичай використовується більш «серйозна» база даних, наприклад MySQL або PostgreSQL.

Якщо ви використовуєте SQLite, вам нічого не потрібно створювати заздалегідь - файл бази даних (за замовчуванням db.sqlite3) створений автоматично при запуску першої команди міграції.

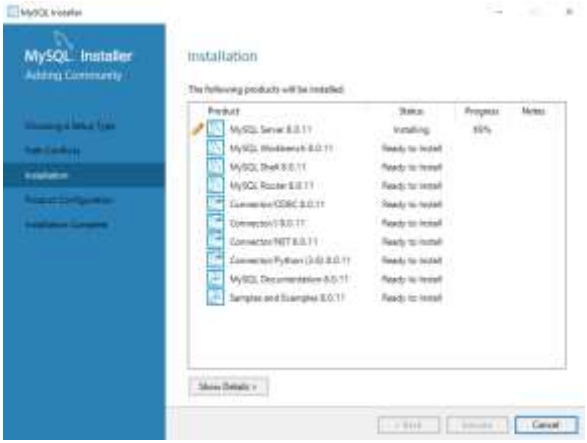


Як уже зазначалось, SQLite підходить для навчальних цілей і не використовується на продакшені.

Для того щоб використовувати іншу «робочу» базу даних *Цей крок можна пропустити і використовувати SQLite.*

Наприклад, установка і налагодження MySQL під Django включає наступні кроки:

1. (якщо не встановлено) Встановіть MySQL або PostgreSQL сервер, при цьому обов'язково запамятайте root пароль. Цей етап при роботі під ОС Windows може викликати певні труднощі.



2. Створіть базу даних для нового сайту та надайте доступ до неї. Знаходячись в директорії, куди проінстальований MySQL Server введіть команду mysql –u root –p і після введення рут-пароля повинні побачити запрошення до вводу sql команд:

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u root -p
Enter password: *******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 8.0.11 MySQL Community Server - GPL

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Створіть базу даних *mysite_db:*

create database mysite_db;

та надайте доступ до неї користувачу, якого вкажете в налаштуваннях (settings.py) сайту:

create user ‘mysite_usr’@’localhost’ identified by ‘mysite_pass’;

grant all privileges on mysite_db.* to ‘mysite_usr’@’localhost’ with grant option;

Поміняйте наступні ключі в елементі ‘default’ настройки DATABASES, щоб вони відповідали налаштуванням підключення до вашої базі даних:

ENGINE - один з

‘django.db.backends.sqlite3’
‘django.db.backends.postgresql’
‘django.db.backends.mysql’
‘django.db.backends.oracle’.
також доступні інші.

NAME - назва вашої бази даних. Якщо ви використовуєте SQLite, база даних буде файлів на вашому комп’ютері, в цьому

випадку NAME містить абсолютний шлях, включаючи ім'я, до цього файлу. Значення за замовчуванням, os.path.join (BASE_DIR, 'db.sqlite3'), створить файл в каталозі вашого проекту.

Якщо ви використовуєте не SQLite, вам необхідно вказати додатково USER, PASSWORD.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'mysite_db',
        'USER': 'mysite_usr',
        'PASSWORD': 'mysite_pass',
    }
}
```

Також зверніть увагу, що користувач бази даних з mysite / settings.py має права створювати базу даних («create database»). Це дозволить автоматично створювати тестову базу даних.

Також зверніть увагу на налаштування INSTALLED_APPS на початку файлу. Вона містить назви всіх додатків Django, які активовані у вашому проекті. Додатки можуть використовуватися на різних проектах, ви можете створити пакет, поширити його і дозволити іншим використовувати його на своїх проектах.

За замовчуванням, INSTALLED_APPS містить наступні додаток, які надаються Django:

django.contrib.admin - інтерфейс адміністратора.

django.contrib.auth - система авторизації.

django.contrib.contenttypes - фреймверк типів даних.

django.contrib.sessions - фреймверк сесії.

django.contrib.messages - фреймверк повідомлень.

django.contrib.staticfiles - фреймверк для роботи зі статичними файлами.

Ці додатки включені за замовчуванням для покриття основних завдань.

Деякі додатки використовують мінімум одну таблицю в базі даних, тому їх створити перед тим, як використовувати, так

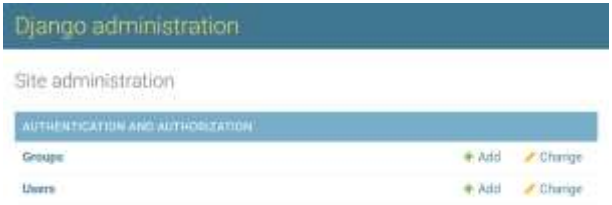
само створюються автоматично при запуску першої команди міграції. Якщо ви змінили базу даних з SQLite на MySQL, цю команду потрібно запустити знову для створення схеми, та суперюзера:

(якщо деактивували віртуальне оточення чи закривали термінал, активуйте знову знаходячись в кореневому каталозі D:\work\myblog>myvenv\Scripts\activate)

python manage.py migrate
python manage.py createsuperuser

В браузері перейдіть за адресою http://127.0.0.1:8000/admin/, повинно з'явитися вікно входу в адмін-панель. Сюди необхідно ввести дані, які ви вказали при створенні суперюзера. Коли ви успішно увійдете до системи, перед вами відкриється головна сторінка адміністративної панелі, через яку ви можете управляти вашими додатками, редагуючи існуючі записи в базі даних або генеруючи нові.

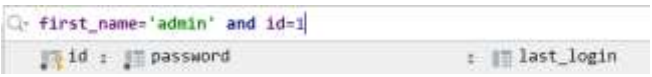
Зверніть увагу, що, незважаючи на те, що вами ще не було створено ніяких моделей і записів в базі даних, в адмінці вже є розділ аутентифікації, це одна з найбільш зручних особливостей Django.



Розкривши вкладку Users, побачите запис з даними користувача, який був створений з консолі як суперюзер.

Запис про цього користувача можна побачити також в базі даних.

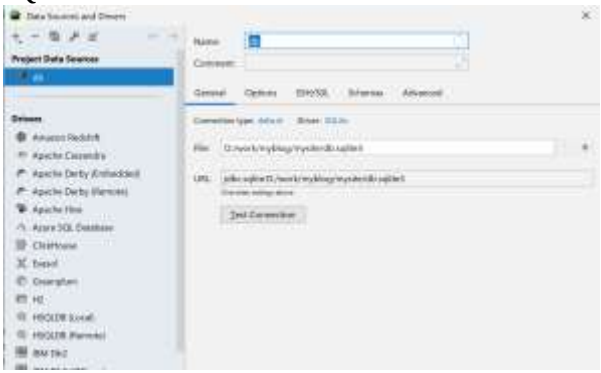
Крім того записи тут можна редагувати та фільтрувати використовуючи рядочок <filter criteria>:



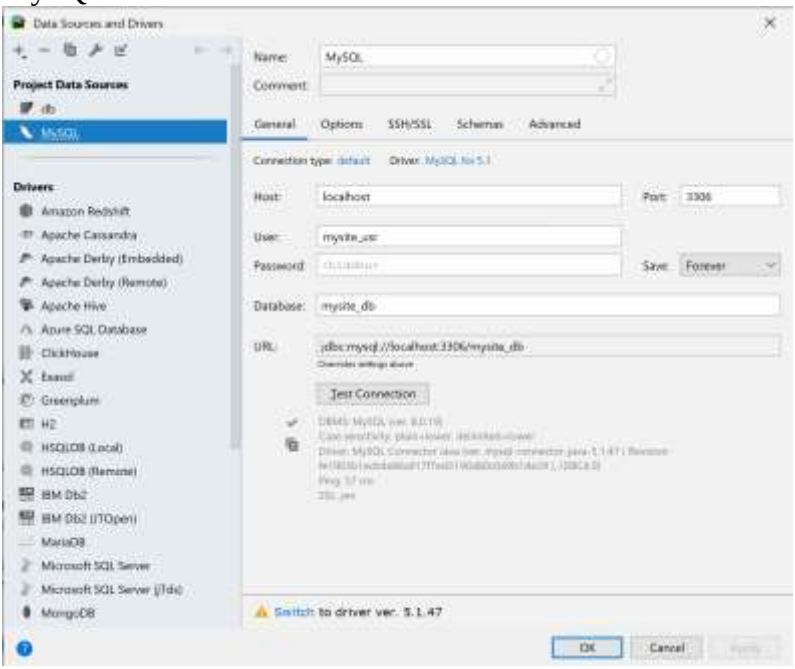
Для налаштування підключення до бази даних в PyCharm відкрийте вкладку Database, натисніть  та підключіть в

налаштуваннях свою (SQLite3 або MySQL) базу даних, як показано нижче:

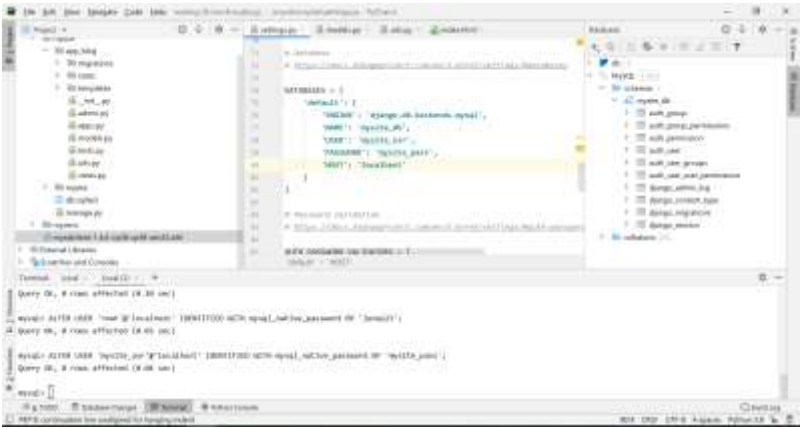
SQLite:



MySQL



У разі успішного підключення в правій частині (вкладка DataBase) побачите схему бази даних.



Завдання:

Додайте декілька користувачів за допомогою панелі адміністратора та переконайтесь у тому, що записи створені в базі даних. Дослідіть схему бази даних, проаналізуйте наявні таблиці. Закомітьте зміни в репозиторій на GitHub та додайте посилання на репозиторій до звіту.

8.6. Контрольні запитання.

8.6.1. Що таке аплікація?

8.6.2. Яка архітектура використовується в Django Fraimwork?

8.6.3. Для чого призначений файл urls?

8.6.4. Для чого призначений файл views?

8.6.5. Де необхідно вказувати налаштування бази даних для проекту?

Література:

1. Лутц М. Программирование на Python, том I, 4-е издание.– Пер. сангл. – СПб.: Символ-Плюс, 2011.– 992с.
2. Документація Django [Електронний ресурс] / Режим доступу : https://djbook.ru/rel3.0

Лабораторна робота №9
Створення моделей та робота з ORM

9.1. Мета роботи

Здобуття практичних навичок використання об'єктно-реляційного відношення (ORM).

9.2. Теоретичні відомості

ORM

Щоб уникати роботи напряму із мовою SQL розробники створили такий собі "місток" між базою даних та об'єктами основної мови програмування. Таким чином кожного разу, коли потрібно отримати чи змінити дані в базі, програміст працює в межах своєї основної мови програмування на рівні об'єктів.

Такий місток має назву ORM (Object Relational Mapping - Об'єктно Реляційне Відображення). Іншими словами - це співставлення записів в таблиці бази даних із концепціями об'єктно-орієнтованої мови програмування, а саме з об'єктом:

• якщо нам потрібно додати новий запис в таблиці - ми створюємо новий об'єкт з класу відповідного даній таблиці;

• якщо ми хочемо оновити існуючий рядок таблиці - через клас типу, що відповідає даній таблиці, ми отримуємо об'єкт даного рядка і змінюємо його атрибути;

• якщо ми хочемо видалити існуючий рядок таблиці - викликаємо відповідний метод на об'єкті, що відповідає даному рядку.

Вище описане співставлення - це одна із реалізацій ORM. Зокрема в Django ORM працює саме таким чином.

Django фреймворк має свою власну ORM-ку. Вона вважається однією із найпотужніших в світі Python з точки зору функціоналу та простоти роботи із нею.

Іншою ORM системою є SQLAlchemy, яка є незалежна від фреймворків і її можна використовувати будь-де, де у вас є справа з мовою програмування Python та реляційною базою даних.

Основа Django підходу є робота з Python класами, які називаємо моделями. Моделі зазвичай визначаються в додатку models.py. Вони реалізуються як підкласи django.db.models.Model, і можуть включати поля, методи і метадані.

Кожен клас повинен унаслідуватись від базового класу “Model” та містити набір атрибутів, кожен з яких описує поле таблиці. Таким чином клас моделі описує структуру однієї таблиці в базі даних, а об’єкт даного класу відображає один запис (рядок) таблиці.

Ось швидкий приклад Django моделі, що описує таблицю із продуктами:

```
from django.db import models


class Product(models.Model):
    title = models.CharField(max_length=256, blank=False,
verbose_name="Product Title")
    price = models.IntegerField(blank=False, default=0,
verbose_name="Product Price")
```

Усе необхідне для роботи з Django моделями лежить в пакеті “django.db”. Визначивши клас Product, який унаслідується від “Model”, ми отримуємо цілий набір функціоналу з доступу та управління таблицею продуктів. Таким чином після синхронізації даної моделі Product з базою даних, в базі даних ми отримаємо нову таблицю для продуктів. В цій таблиці, окрім поля первинного ключа (ID, Primary Key), ми матимемо стрічкове поле (CharField) “title” (назва продукту) та ціле число (IntegerField) “price” (ціна продукту).

При першому завантаженні нашого класу Product та синхронізації бази даних Django створить таблицю в базі даних. При цьому Django ORM підготує та запустить подібний до наступного SQL запит:

```
CREATE TABLE demoapp_product (
   "id" integer AUTO_INCREMENT NOT NULL PRIMARY KEY,
```

```
    "title" VARCHAR(256) NOT NULL,
    "price" integer DEFAULT 0 NOT NULL
);
```

В даному SQL запиті назва таблиці складається з назви аплікації (demoapp) та назви класу моделі продукту (product) розділеними нижнім підкресленням.

Так по-замовчуванню Django ORM називає таблиці моделей.

Поле “id” є цілим числом, що збільшується автоматично на одиницю з кожним новим рядком (записом) в таблиці. Воно не може бути NULL, що означає порожнє значення. PRIMARY KEY означає, що дане поле є унікальним і по ньому можна однозначно отримувати рядок з таблиці.

Поле “title” є стрічкою з максимальною довжиною в 256 символів і також не може бути порожнім в базі.

Останнім полем буде створено поле “price”, яке також є цілим числом. Якщо при додаванні нового запису в дану таблицю не буде передане поле ціни, тоді воно автоматичо встановиться у нуль. Це завдяки параметру “DEFAULT 0”.

Django ORM надає цілий ряд типів полів, які перевищують по кількості типи в базі даних. Кожне із полів задекларованих в класі моделі має цілий ряд параметрів для конфігурації. Частина цих параметрів безпосередньо впливає на структуру таблиці в базі (default, сам тип поля, null, max_length і т.п.), інша ж частина має відношення суто до Django функціоналу як от адміністративної частини (blank, verbose_name, і т.п.).

Поля використовуються не лише для опису структури таблиці в базі даних, але й для формування форм та полів (віджетів) всередині цих форм. Форми використовуються в Django адміністративній частині для управління даними в базі.

9.3. Програма роботи

9.3.1. Вивчити основні принципи роботи з моделями та базою даних в Django.

9.3.2. Зареєструвати моделі в адмін-панелі.

9.3.3. Виконати міграції бази даних.

9.3.4. Здійснити початкове наповнення бази даних блогу.

9.4. Обладнання та програмне забезпечення

9.4.1. Персональний комп'ютер.

9.4.2. Інтерпритатор Python встановлений на ПК

9.4.3. Середовище розробки програмного забезпечення PyCharm Community Edition (чи інше IDE, наприклад Sublime Text, Geany).

9.4.4. Web-фреймворк Django.

9.5. Порядок виконання роботи і опрацювання результатів

Створення моделей

Тепер створимо ваші моделі - по суті структуру вашої бази даних з додатковими мета-даними.

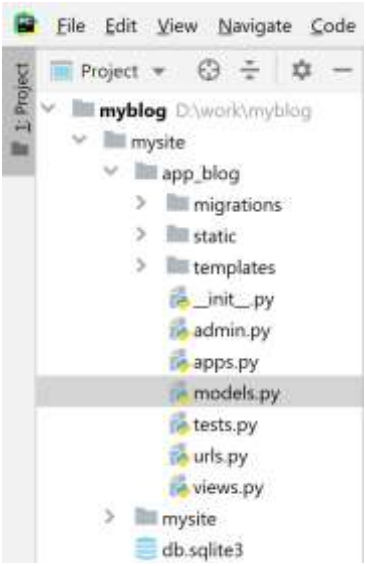
Модель - це основне джерело даних. Він містить набір полів і поведінку даних, які ви зберігаєте. Django дотримується принципу DRY. Сенс в тому, щоб визначати моделі в одному місці.

Створимо три моделі для майбутнього блогу:

Категорія (Category) – будемо створювати публікації, при цьому сортуючи їх за категоріями, які можна буде додавати за допомогою панелі адміністратора

Стаття (Article) – власне сама публікація, яка буде містити заголовок та текст, дату публікації та слаг (рядок-ідентифікатор, у веб розробці використовується в URL, а також в запитах до бази даних), можливість додати публікацію на головну сторінку сайту, категорію.

Зображення (ArticleImage) – зображення буде «відноситись» до певної публікації, але оскільки ми додамо можливість до однієї публікації додавати не одне а декілька (або жодного, тобто наперед не визначену кількість), то виносимо зображення в окремий клас із відношенням «один-до-багатьох» - одна стаття –багато зображень (ForeignKey).



Вміст файлу models.py:

```
# -*- coding: utf-8 -*-

from django.utils import timezone

from django.db import models
from django.urls import reverse

# Create your models here.
# mysite /urls.py


class Category(models.Model):
    category = models.CharField(u'Категорія',
max_length=250, help_text=u'Максимум 250 символів')

    class Meta:
        verbose_name = u'Категорія для новини'
        verbose_name_plural = u'Категорії для новин'

    def __str__(self):
        return self.category
```

```
class Article(models.Model):
    title = models.CharField(u'Заголовок', max_length=250,
                             help_text=u'Максимум 250
символів')
    description = models.TextField(blank=True,
                                   verbose_name=u'Опис')
    pub_date = models.DateTimeField(u'Дата публікації',
                                    default=timezone.now)
    slug = models.SlugField(u'Слаг',
                            unique_for_date='pub_date')

    main_page = models.BooleanField(u'Головна',
                                    default=False,
                                    help_text=u'Показувати')
    category = models.ForeignKey(Category,
                                 related_name='news',
                                 blank=True,
                                 null=True,
                                 verbose_name=u'Категорія',
                                 on_delete=models.CASCADE)
    objects = models.Manager()

    class Meta:
        ordering = ['-pub_date']
        verbose_name = u'Стаття'
        verbose_name_plural = u'Статті'

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        try:
            url = reverse('news-detail',
                          kwargs={
                    'year': self.pub_date.strftime("%Y"),
                    'month': self.pub_date.strftime("%m"),
                    'day': self.pub_date.strftime("%d"),
                    'slug': self.slug,
            })
        except:
            url = "/"
        return url
```

```
class ArticleImage(models.Model):
    article = models.ForeignKey(Article,
                                verbose_name=u'Стаття',
                                related_name='images',
                                on_delete=models.CASCADE)
    image = models.ImageField(u'Фото', upload_to='photos')
    title = models.CharField(u'Заголовок', max_length=250,
                             help_text=u'Максимум 250
символів', blank=True)

    class Meta:
        verbose_name = u'Фото для статті'
        verbose_name_plural = u'Фото для статті'

    def __str__(self):
        return self.title

    @property
    def filename(self):
        return self.image.name.rsplit('/', 1)[-1]
```

- utf-8 - це заголовок про кодування файлу, щоб можна було використовувати кирилицю в модулі;
- import models - модуль models дає нам доступ до Django ORM: базових класів моделей, полів, констант та конфігурації пов'язаної із базою даних; унаслідуємось обов'язково від Model класу; завдяки йому ми матимемо усю силу Django ORM використовуючи наші класи;
- *CharField* це рядкове поле, що в базі MySQL відповідає полю VARCHAR; назва атрибута буде використана для назви поля в таблиці бази даних; *max_length* - це обов'язковий параметр поля *CharField*, який визначає максимальну довжину стрічки дозволеної у даному полі; blank - на рівні форм управління об'єктами даної моделі, дане поле буде обов'язковим до заповнення; *verbose_name* - вказує на рядок, під яким представляти дане поле на користувацькому інтерфейсі; якщо даний параметр не вказаний, використовуватиметься назва атрибуту поля; *default* - корисний параметр у випадку, якщо поле не є обов'язковим; якщо на

формі управління об’єкта моделі поле залишене не заповненим, тоді Django ORM автоматично передасть значення із параметра default в якості значення поля в базі даних;

- поле типу *DateTimeField*, що вказує на дату; дане поле відповідає типу DATE в базі даних MySQL.
- навідміну від *CharField*, *TextField* дозволяє містити багаторядковий текст і не потребує фіксованої довжини; на формах дане поле представлене тегом textarea, який є багаторядковим полем для вводу тексту.
- image - дане поле є типом *ImageField*; воно міститиме для нас зображення студента; насправді сам файл зображення Django не зберігає в базі, а у файловій системі в папці MEDIA_ROOT (далі буде більше деталей про дану папку), а в базі зберігається лише назва завантаженого файла;
- *null vs blank:* досить часто плутають дані два параметри полів в моделях. Насправді вони мають різну мету. null конфігурує базу даних вказуючи на те, чи може бути поле таблиці порожнім. В той час як blank вказує Django на те, чи поле на формі управління моделлю буде обов’язковим чи ні. Зазвичай вони працюють в парі і якщо blank є True (поле необов’язкове на формі), тоді null також є True (поле може бути порожнім в базі даних).

Детально про типи полів моделей https://djbook.ru/rel3.0/topics/db/models.html

- *ImageField* Django ORM не зберігає файлів в базі. Це може знизити швидкодію бази даних. Натомість бінарні файли зберігаються у файловій системі.

В налаштуваннях проекту (модуль settings.py) потрібно вказати абсолютний шлях до папки, де будуть зберігатись усі файли, що відносяться до даних аплікації. Ця змінна називається “MEDIA_ROOT”. Подібним чином треба налаштувати URL адресу, яка обслуговуватиме дані зображення в браузері.

Додаємо наступні рядки до **settings.py** модуля:

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/
```

```
STATIC_URL = ‘/static/’

STATIC_ROOT = os.path.join(BASE_DIR, ‘static’)

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, ‘static’),
]

MEDIA_URL = ‘/media/’
MEDIA_ROOT = os.path.join(BASE_DIR, ‘media’)
```

Директорія media буде додана автоматично. Таким чином в браузері під адресою “http://localhost:8000/media/” можна буде отримувати файли, що лежатимуть у медіа директорії. А папку із файлами у файловій системі ми покладемо на рівень вище, ніж корінь проекту, в папку “media”.

Також для роботи поля ImageField необхідна Python бібліотека **Pillow** (http://pillow.readthedocs.org/en/latest/). Дана бібліотека дозволяє працювати із зображеннями та модифікувати їх з допомогою Python коду. Інсталюйте її

python -m pip install Pillow

Кожного разу коли інсталюєте будь-який пакет, його потрібно додати в requirements.txt (див. лаб. №7).

Зверніть увагу на метод *get_absolute_url*, який ми будемо пізніше використовувати для формування url-адреси конкретної публікації, наприклад

/news/2020/02/25/bezpeka-kiberprostoru/

Останньою частиною цієї адреси є слаг.

Метод __str__ (в Python 2.x __unicode__) визначає, як буде відображатися об’єкт при виведенні через print, в адмінці, при використанні в шаблоні.

Для створення таблиць в базі даних, що будуть відповідати оголошеним моделям, в консолі (під віртуальним оточенням) запускаємо команду

python mysite/manage.py makemigrations

Результатом виконання повинно бути повідомлення
Migrations for 'app_blog':
mysite\app_blog\migrations\0001_initial.py
- Create model Article
- Create model Category
- Create model ArticleImage
- Add field category to article

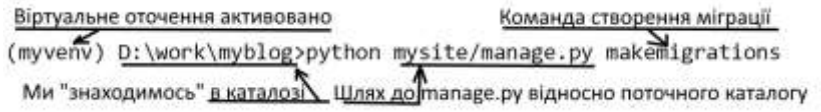
Та команду

python mysite/manage.py migrate

Результатом виконання повинно бути повідомлення
Operations to perform:
Apply all migrations: admin, app_blog, auth, contenttypes, sessions
Running migrations:
Applying app_blog.0001_initial... OK

Якщо при цьому виникли помилки:

1. Зверніть увагу на правильність запуску команд



2. Переконайтесь в правильності налаштування бази даних. Якщо створювали та підключали MySQL базу даних, можливо виникла проблема з доступами користувача або не встановлений mysqlclient.

Заінсталюйте mysqlclient
- pip install wheel
- https://www.lfd.uci.edu/~gohlke/pythonlibs/#mysqlclient скачати необхідну версію mysqlclient (наприклад

mysqlclient-1.4.6-cp38-cp38-win32) в ту директорію, звідки робиться pip

- pip install mysqlclient-1.4.6-cp38-cp38-win32.whl

Змініть пароль для користувача ‘mysite_usr’ на той, що вказаний в settings.py:

- Перейдіть в C:\Program Files\MySQL\MySQL Server 8.0\bin,
- mysql -u root –p
- ALTER USER ‘mysite_usr’@’localhost’ IDENTIFIED WITH mysql_native_password BY ‘mysite_pass’;

Реєстрація моделей

Наступним кроком є реєстрація створених моделей в панелі адміністратора, без цього кроку ви не зможете наповнювати базу даних даними через адмін панель. Тобто для того, щоб модель з’явилась в адмінці, необхідно оновити модуль admin.py в корені аплікації наступним рядком (в найпростішому випадку):

```
from django.contrib import admin
from .models import Article

# Register your models here.
admin.site.register(Article)
```

Функція register повідомляє Django про нашу модель і просить його додати її до адмін частини.

Щоб отримати **user-friendly** ім’я моделі в адмін інтерфейсі Django всюди, де є посилання на модель, використовується вкладений клас Meta всередині класу моделі. В класі Meta ми визначили два атрибути: verbose_name та verbose_name_plural. Перший визначає рядок для використання в Django адмінці, щоб позначати модель, наприклад ‘Стаття’. Другий - модель у формі множини.

Відкрийте файл admin.py app_blog та додайте реєстрацію моделей:

```
# -*- coding: utf-8 -*-

from django.contrib import admin
from django.shortcuts import get_object_or_404

from .models import Article, ArticleImage, Category
from .forms import ArticleImageForm


class CategoryAdmin(admin.ModelAdmin):
    list_display = (‘category’,)
    fieldsets = (
        (‘‘, {
            ‘fields’: (‘category’, ),
        }),
    )
admin.site.register(Category, CategoryAdmin)


class ArticleImageInline(admin.TabularInline):
    model = ArticleImage
    form = ArticleImageForm
    extra = 0

    fieldsets = (
        (‘‘, {
            ‘fields’: (‘title’, ‘image’,),
        }),
    )
class ArticleAdmin(admin.ModelAdmin):
    list_display = (‘title’, ‘pub_date’, ‘slug’, ‘main_page’)
    inlines = [ArticleImageInline]
    multiupload_form = True
    multiupload_list = False
    prepopulated_fields = {‘slug’: (‘title’,)}
    raw_id_fields = (‘category’,)
    fieldsets = (
        (‘‘, {
            ‘fields’: (‘pub_date’, ‘title’, ‘description’,
‘main_page’),
```

```
        }),
        ((u'Додатково'), {
            'classes': ('grp-collapse grp-closed',),
            'fields': ('slug',),
        }),
    )

    def delete_file(self, pk, request):
        '''Delete an image.'''

        obj = get_object_or_404(ArticleImage, pk=pk)
        return obj.delete()


admin.site.register(Article, ArticleAdmin)
```

Зверніть увагу на рядок

```
from .forms import ArticleImageForm
```

Якщо на даному етапі ви спробуєте запустити сервер, то отрмаєте помилку. Справа в тому, що ми намагаємось імпортувати клас ArticleImageForm з файла forms.py поточної директорії, якого в проекті немає.

Додамо файл forms.py в директорію app_blog (де знаходиться models.py) із таким вмістом:

```
# -*- coding: utf-8 -*-

from django import forms

from .models import ArticleImage


class ArticleImageForm(forms.ModelForm):
    image = forms.ImageField(
        widget=forms.ClearableFileInput(attrs={'multiple':
True}))

    class Meta:
        model = ArticleImage
        fields = '__all__'
```

Призначення його – створення форми для додавання файлів із зображеннями. При цьому використовується віджет для передачі даних. Детальніше про віджети https://djbook.ru/rel3.0/ref/forms/widgets.html

Після того як даний файл додано, запустіть сервер та перейдіть на сторінку адміністратора (http://127.0.0.1:8000/admin).

Побачите нову секцію APP_BLOG, в якій з'явились дві зареєстровані моделі Category – «Категорії для новин» та Article – «Статті». Реєстрації моделі в адмінці відповідають команди

admin.site.register(Модель, МодельAdmin)

яка звязує форму в адмін панелі з відповідною моделлю. Оскільки в файлі admin.py таких команд дві, відповідно м бачимо дві зареєстровані моделі. Імена моделей, що відображаються в адмінці, відповідають значенням *verbose_name_plural* із *class Meta* відповідних моделей.

Щодо третьої моделі ArticleImage, вона зареєстрована як

inlines = [ArticleImageInline],

тобто додавання зображень буде «прив'язане» до додавання публікації.

Створення міграцій

Змінимо моделі аплікейшина, наприклад *help_text* для поля *main_page* та *related_name* для поля *category* моделі *Article*:

```
 main_page = models.BooleanField(u'Головна', default=False,
    help_text=u'Показувати на головній сторінці')
category = models.ForeignKey(Category,
    related_name='articles', blank=True, null=True,
    verbose_name=u'Категорія',on_delete=models.CASCADE)
```

та *verbose_name* і *verbose_name_plural* та додамо *slug* для моделі *Category*:

```
class Category(models.Model):
    category = models.CharField(u'Категорія',
         max_length=250, help_text=u'Максимум 250 символів')
    slug = models.SlugField(u'Слаг')
```

```
    class Meta:
        verbose_name = u'Категорія для публікації'
        verbose_name_plural = u'Категорії для публікацій'

    def __str__(self):
        return self.category
```

Оскільки наші моделі відповідають схемі бази даних, після того як в файл models.py вносяться які-небуть зміни, необхідно новити схему командою

python mysite/manage.py makemigrations

Після цього побачите список змін, які необхідно застосувати до таблиць бази даних

Migrations for 'app_blog':
mysite\app_blog\migrations\0002_auto_20200309_1626.py
- Change Meta options on category
- Alter field category on article
- Alter field main_page on article
- Add field slug to category

Щоб їх застосувати виконуємо команду

python mysite/manage.py migrate

Якщо ваші зміни не є суперечливими з поточною схемою, база не «поламана», ви повинні побачити

Operations to perform:
Apply all migrations: admin, app_blog, auth, contenttypes, sessions
Running migrations:
Applying app_blog.0002_auto_20200309_1626... OK

І в директорії migrations вашого аплікейшина app_blog з'явиться файл міграції. Перший 0001_ initial.py був створений при першому виконанні команд makemigrations/migrate. Видаляти ці файли **не можна** (якщо ви не «відкотили» перед цим міграцію).

Перезапустіть сервер командою runserver та перейдіть на сторінку адміністратора. Побачите оновлену назву для категорії

Завдання: Модифікуйте admin.py таким чином, щоб додавати слаг для категорії, використовуючи автозаповнення на основі назви категорії (prepopulated_fields = {‘slug’: (‘category’,)}). Додайте декілька категорій для вашого майбутнього блогу та створіть декілька публкації для кожної з категорій із зображеннями (одне, декілька, або зовсім без зображень), видаліть декілька записів і переконайтесь, що вони видалились із бази даних. Закомітьте зміни в репозиторій на GitHub та додайте посилання на репозиторій до звіту.

9.6. Контрольні запитання.

9.6.1 Дати визначення об’єктній реляційній проекції?
9.6.2 Що означає аргумент default для поля?
9.6.3 Яке призначення файла admin.py?
9.6.4 Яке призначення команди makemigrations?

Література:

1. Лутц М. Программирование на Python, том I, 4-е издание.– Пер. сангл. – СПб.: Символ-Плюс, 2011.– 992с.
2. Документація Django [Електронний ресурс] / Режим доступу : https://djbook.ru/rel3.0

Лабораторна робота №10
Розробка серверної частини персонального блогу. Модульне тестування веб-додатку

10.1. Мета роботи

Познайомитись з принципами написання відображень та створення шаблонів для створення динамічних веб-сторінок засобами Django Frimwork. Познайомитись з базовими принципами тестування веб-аплікацій.

10.2. Теоретичні відомості

Диспетчер URL

Диспетчер URL та **URLconf** (конфігурація URL) є основними частинами Django застосунку. Спочатку це може здаватися заплутаним.

Проект може містити багато **urls.py** розподілених між застосунками. Але Django потребує **urls.py**, щоб використовувати його як відправну точку. Цей спеціальний **urls.py** називається **root URLconf**. Він визначається у файлі **settings.py**.

```
ROOT_URLCONF = 'mysite.urls'
```

Він вже налаштований, тому тут нічого не потрібно змінювати.

Коли Django отримує запит, він починає шукати відповідність в URLconf проекту. Він починає з першого вводу змінної urlpatterns і перевіряє запитувану URL-адресу навпроти кожної введеної url.

Якщо Django знайде відповідність, він передасть запит до функції представлення, яка є другим параметром url. Порядок в urlpatterns має значення, тому що Django припиняє пошук, як тільки знаходить відповідність. Тепер, якщо Django не знайде відповідності в URLconf, він викличе виняток **404**, що є кодом

помилки для **Page Not Found**, тобто запитувана сторінка не була знайдена.

Базові URL-адреси дуже просто створювати. Це лише питання відповідності рядків. Наприклад, скажімо, ми захотіли створити сторінку «about», її можна було б визначити наступним чином:

```
from django.conf.urls import url
from app_blog import views

urlpatterns = [
    url(r'^$', views.home, name='home'),
    url(r'^about/$', views.about, name='about'),
]
```

Ми також можемо створити більш глибокі URL-структури:

```
from django.conf.urls import url
from app_blog  import views

urlpatterns = [
    url(r'^$', views.home, name='home'),
    url(r'^about/$', views.about, name='about'),
    url(r'^about/company/$', views.about_company,
name='about_company'),
]
```

Більш просунуте використання маршрутизації URL-адрес досягається завдяки використанню регулярного виразу для відповідності визначеним типам даних та створення динамічних URL-адрес.

Наприклад, щоб створити сторінку профілю, як це роблять багато сервісів, таких як facebook.com/codeguida або twitter.com/codeguida, де «codeguida» — ім'я користувача, ми можемо зробити наступне:

```
from django.conf.urls import url
from app_blog import views

urlpatterns = [
    url(r'^$', views.home, name='home'),
    url(r'^(?P<username>[\w.@+-]+)/$', views.user_profile,
```

```
name='user_profile'),
]
```

Це відповідатиме всім дійсним іменам користувача для Django моделі User.

Тепер зверніть увагу на те, що наведений вище приклад є дуже *дозвільною (permissive)* URL-адресою. Це означає, що він відповідатиме багатьом шаблонам URL, оскільки визначається в кореневому каталозі URL без префіксу як, наприклад **/profile/<username>/**. У цьому випадку, якщо ми хочемо визначити URL-адресу з ім'ям **/ about /**, нам слід визначити її *перед* шаблоном URL користувача:

```
from django.conf.urls import url
from app_blog import views

urlpatterns = [
    url(r'^$', views.home, name='home'),
    url(r'^about/$', views.about, name='about'),
    url(r'^(?P<username>[\w.@+-]+)/$', views.user_profile,
name='user_profile'),
]
```

Якщо сторінка «about» була визначена *після* шаблону URL-адреси користувача, Django ніколи не знайде його, тому що слово «about» буде відповідати регулярному виразу з ім'ям користувача, а представлення user_profile буде оброблене замість функції представлення about.

Існують деякі побічні ефекти. Наприклад, відтепер нам слід розглядати «about» як заборонене ім'я користувача, оскільки якщо користувач вибрав «about» в якості імені, він ніколи не побачить свою сторінку профілю.

До речі, якщо ви хочете створити класні URL для профілів користувачів, найлегшим рішенням для уникання суперечностей між URL, буде додавання префіксу, на кшталт **/u/username/**, або **/@username**, де @ є префіксом. Якщо ви не бажаєте взагалі використовувати префікс, подумайте про те, щоб використовувати список заборонених імен.

Ідея такого роду URL-маршрутизації полягає у створенні динамічних сторінок, де частина URL-адреси буде

використовуватися як ідентифікатор певного ресурсу, який, у свою чергу, буде використовуватися для складання сторінки. Цей ідентифікатор може бути, наприклад, цілочисельним ID або рядком.

Основні елементи синтаксису регуляних виразів

Деякі базові елементи регуляних виразів, які можна використовувати для визначення адрес URL:

^ (Початок адреси)

$ (Кінець адреси)

+ (1 і більше символів)

? (0 або 1 символ)

{N} (n символів)

{N, m} (від n до m символів)

. (Будь-який символ)

\ D + (одна або кілька цифр)

\ D + (одна або кілька НЕ цифр)

\ W + (один або кілька буквених символів)

Адреса	Запит
r'^$'	http://127.0.0.1/ (початкова сторінка)
r'^about'	http://127.0.0.1/about/ http://127.0.0.1/about/contact
r'^about\contact'	http://127.0.0.1/about/contact
r'^products/\d+/'	http://127.0.0.1/products/23/ http://127.0.0.1/products/6459/abc Але не відповідає http://127.0.0.1/products/abc/
r'^products/\D+/'	http://127.0.0.1/products/abc/ http://127.0.0.1/products/abc/123 Але не віповідає http://127.0.0.1/products/123/ http://127.0.0.1/products/123/abc
r'^products/phones|tablets/'	http://127.0.0.1/products/phones/1 http://127.0.0.1/products/tablets/ Але не віповідає http://127.0.0.1/products/clothes/

r'^products/\w+'	http://127.0.0.1/abc/ http://127.0.0.1/123/ Але не віповідає http://127.0.0.1/abc-123
r'^products/[-\w]+/'	http://127.0.0.1/abc-123
r'^products/[A-Z]{2}/'	http://127.0.0.1/UA Але не віповідає http://127.0.0.1/Ua http://127.0.0.1/UAH

Відображення

Центральним моментом будь-якого веб-додатку є обробка запиту, який відправляє користувач. В Django за обробку запиту відповідають представлення або views. По суті представлення представляють функції обробки, які приймають дані запиту у вигляді об'єкта request і генерують деякий результат, який потім відправляється користувачеві.

За замовчуванням представлення розміщуються в додатку в файлі views.py.

Відображення - *view* - це місце, в якому закладена "логіка" програми. Воно запитує інформацію з моделі і передає його до шаблону. Відображення - це просто методи Python.

Шаблони

Будучи веб фреймверком, Django дозволяє динамічно генерувати HTML. Найпоширеніший підхід - використання шаблонів. Шаблони містять статичний HTML і динамічні дані, рендеринг яких описаний спеціальний синтаксис. Проект Django може використовувати один або кілька механізмів створення шаблонів (або жодного, якщо ви не використовуєте шаблони). Django надає бекенд для власної системи шаблонів, яка називається - мова шаблонів Django (Django template language, DTL), і популярного альтернативного шаблонізатора Jinja2. Сторонні додатки можуть надавати бекенд і для інших систем шаблонів.

Django надає стандартний API для завантаження і

рендеринга шаблонівЗавантаження включає в себе пошук шаблону за назвою і попередню обробку, зазвичай виконується завантаження шаблону в пам'ять. Візуалізація означає передачу даних контексту в шаблон і повернення рядка з результатом.

Мова шаблонів Django - власна система шаблонів Django. Підтримка шаблонів і вбудована система шаблонів Django знаходяться в одному пакеті django.template.

В HTML, не можна помістити Python код, тому що браузери не зрозуміють його. Вони знають лише HTML. HTML більшою мірою є статичною, в той час як Python є більш динамічною мовою.

Шаблонні теги Django дозволяють передавати Python-подібні речі в HTML, таким чином можна розробляти динамічні веб-сайти.

Теги

Теги дозволяють додавати довільну логіку в шаблон.

Наприклад, теги можуть виводити текст, додавати логічні оператори, такі як "if" або "for", отримувати вміст з бази даних, або надавати доступ до інших тегами.

Теги виділяються {% i%}, наприклад:{% csrf_token%}

Більшість тегів приймають аргументи:
{% cycle 'odd' 'even'%}

Деякі теги вимагають закриваючий тег:
{% wf user %} Hello, {{ user.username }}. {% Endif %}

Ознайомтеся зі списком всіх вбудованих тегів і з керівництвом по створенню тегів можна на сторінці Django-документації https://djbook.ru/rel3.0/ref/templates/builtins.html.

Фільтри

Фільтри перетворюють змінні і аргументи тегів. Можуть виглядати таким чином:

{{ 'django'| title }}

Для контексту { ‘django’: ‘the web framework for perfectionists with deadlines’} цей шаблон виведе:

The Web Framework For Perfectionists With Deadlines

Тобто у форматі заголовка – перша літера кожного слова у верхньому регістрі.

Деякі фільтри приймають аргументи:

{{My_date | date: "Y-m-d"}}

Робота з формами

Якщо ви плануєте створювати сайти і додатки, який приймають і зберігають дані від користувачів, вам необхідно використовувати форми. Django надає широкий набір інструментів для цього.

Форма в HTML - це набір елементів в <form> ... </ form>, які дозволяють користувачеві вводити текст, вибирати опції, змінювати об’єкти і контороля сторінки, і так далі, а потім відправляти цю інформацію на сервер.

Деякі елементи форми - текстові поля введення і чекбокси - досить прості і вбудовані в HTML. Деякі - досить складні, складаються з діалогів вибору дати, слайдерів і інших контролів, який зазвичай використовують JavaScript і CSS.

Крім <input> елементів форма повинна містити ще дві речі:

- куди: URL, на який будуть відправлені дані
- як: HTTP метод, який має використовувати форма для відправки даних.

Наприклад, форма авторизації Django містить кілька <input> елементів: один з type = "text" для логіна користувача, один з type = "password" для пароля, і один з type = "submit" для кнопки "Log in". Вона також містить кілька прихованих текстових полів, які Django використовує для визначення що робити після авторизації.

Форма також говорить браузеру, що дані повинні відходити на URL, вказаний в атрибуті action тега <form> - / admin / - і для відправки необхідно використовувати HTTP метод, зазначений атрибуті method - post.

При натисканні на елемент <input type = "submit" value = "Log in"> дані будуть відправлені на / admin /.

GET і POST - єдині HTTP методи, які використовуються для форм.

Форма авторизації в Django використовує POST метод. При відправці форми браузер збирає всі дані форми, кодує для відправки, відправляє на сервер і отримує відповідь.

При GET, на відміну від POST, дані збираються в рядок і передаються в URL. URL містить адресу, куди відправляти дані, і дані для відправки. GET і POST зазвичай використовуються для різних дій.

Будь-який запит, який може змінити стан системи - наприклад, який змінює дані в базі даних - повинен використовувати POST. GET повинен використовуватися для запитів, які не впливають на стан системи.

Не слід використовувати GET для форми з паролем, тому що пароль з'явиться в URL, а отже в історії браузера. Також він не підходить для відправки великої кількості даних або бінарних даних, наприклад, зображення. POST використовує додаткові механізми захисту, наприклад, CSRF захист, і надає більше контролю за доступом до даних.

Работа з формами - досить не просте завдання. Візьмемо адмінку Django. Необхідно підготувати для відображення в формі велику кількість даних різного типу, відобразити форму в HTML, створити зручний інтерфейс для редагування, отримати дані на сервері, перевірити і перетворити в потрібний формат, і в кінці зберегти або передати для подальшої обробки.

Форми Django можуть спростити і автоматизувати велику частину цього процесу, і можуть зробити це простіше і надійніше, ніж код, написаний більшістю програмістів.

Django дозволяє:

- підготувати дані для відображення в формі
- створити HTML форми для даних
- отримати і обробити відправлені формою дані

Ви можете написати код, який все це буде робити, але Django може виконати більшу частину самостійно.

Серце всього механізму - **клас Form.** Як і модель в Django, яка описує структуру об'єкта, його поведінка і уявлення, Form описує форму, як вона працює і показується користувачеві.

Як поля моделі представляють поля в базі даних, поля форми представляють HTML <input> елементи. (ModelForm відображає поля моделі у вигляді HTML <input> елементів, використовуючи Form. Використовується в адмінці Django)

Поля форми самі є класами. Вони управляють даними форми і виконують їх перевірку при відправці форми. Наприклад, DateField і FileField працюють з різними даними і виконують різні дії з ними.

Поле форми представлено в браузері HTML "віджетом". Кожен тип поля представлений за замовчуванням певним класом Widget, який можна перевизначити при необхідності.

Створення, обробка та рендеринг форм

При рендеригу об'єкта в Django ми зазвичай:

- отримуємо його в представленні (views) (наприклад, завантажуємо з бази даних)
- передаємо в контекст шаблону
- представляємо у вигляді HTML в шаблоні, використовуючи змінні контексту (теги)

Візуалізація форм відбувається аналогічним чином з деякими відмінностями.

У випадку з моделями, навряд чи нам може знадобиться порожня модель в шаблоні. Для форм же нормально показувати порожню форму для користувача.

Екземпляр моделі, який використовується в представленні, зазвичай завантажується з бази даних. При роботі з формою ми зазвичай створюємо екземпляр форми представлення.

При створенні форми ми може залишити її порожньою, або додати початкові дані, наприклад:

- збереженого раніше об'єкта моделі (для редагування їх в адмінці)
- отримані з інших джерел
- отримання з попередньої відправки форми

Юніт тести

Сайти, в процесі розвитку і розробки, стає все складніше тестувати вручну, тобто переходити по всіх можливих посиланнях з метою виявлення помилок, виконання допустими дій через адмін-панелі т.д. Крім такого тестування, складними стають внутрішні взаємодії між компонентами - внесення невеликої зміни в одній частині додатка впливає на інші. При цьому, щоб все продовжувало працювати потрібно вносити все більше і більше змін і, бажано так, щоб не додавалися нові помилки. Одним із способів, який дозволяє пом'якшити наслідки додавання змін, є впровадження в розробку автоматичного тестування - воно повинно просто і надійно запускатися кожен раз, коли ви вносите зміни в свій код.

Тестування сайту це складне завдання, тому що складається з декількох логічних шарів - від HTTP-запиту і запиту до моделей, до валідації форми і їх обробки, а крім того, рендеринга шаблонів сторінок.

Django надає фреймворк для створення тестів, побудованого на основі ієрархії класів, які, в свою чергу, залежать від стандартної бібліотеки Python unittest. Незважаючи на назву, даний фреймворк підходить і для юніт-, і для інтеграційного тестування. Фреймворк Django додає методи API і інструменти, які допомагають тестувати як веб так і, специфічну для Django, поведінку. Це дозволяє вам імітувати URL-запити, додавання тестових даних, а також проводити перевірку вихідних даних ваших додатків. Крім того, Django надає API (LiveServerTestCase) і інструменти для застосування різних фреймфорков тестування, наприклад ви можете підключити популярний фреймворк Selenium для імітації поведінки користувача в реальному браузері.

Для написання тесту ви повинні успадковуватися від будь-якого з класів тестування Django (або юніттеста) (SimpleTestCase, TransactionTestCase, TestCase, LiveServerTestCase), а потім реалізувати окремі методи перевірки коду (тести це функції-"затвердження", які перевіряють, що результатом виразу є значення True або False,

або що два значення рівні і так далі). Коли ви запускаєте тест, фреймворк виконує відповідні тестові методи в вашому класі-нащадку. Методи тестування запускаються незалежно один від одного, починаючи з методу налаштувань і / або завершуючи методом руйнування (tear-down), визначеному в класі, як показано нижче.

```
class YourTestClass(TestCase):

    def setUp(self):
        # Налаштування запускаются перед кожним тестом
        pass

    def tearDown(self):
        # Очистка після кожного методу
        pass

    def test_something_that_will_pass(self):
        self.assertFalse(False)

    def test_something_that_will_fail(self):
        self.assertTrue(False)
```

Самий відповідний базовий клас для більшості тестів це django.test.TestCase. Цей клас створює чисту базу даних перед запуском своїх методів, а також запускає кожну функцію тестування в його власній транзакції. У даного класу також є тестовий Клієнт, який ви можете використовувати для імітації взаємодії користувача з кодом на рівні відображення.

Потрібно тестувати всі аспекти, що стосуються вашого коду, але не бібліотеки, або функціональність, що надаються Python, або Django.

Наприклад, не потрібно перевіряти той факт, що певне поле моделі, яке наприклад задекларованe як CharField, були насправді збережені в базу даних як CharField, тому що за це відповідає безпосередньо Django.

Стуктура тестів

Django використовує юніт-тестовий модуль - вбудований

"виявляч" тестів, який знаходить тести в поточній робочій директорії, в будь-якому файлі з шаблонною назвою test*.py. Надаючи відповідні імена файлів, ви можете працювати з будь-якою структурою, яка вас влаштовує. Ми рекомендуємо створити пакет для вашого поточного коду і, отже, відокремити файли моделей, відображень, форм і будь-які інші, від коду який буде використовуватися для тестів.

Найпростішим способом запуску всіх тестів є застосування такої команди:

python manage.py test

Таким чином ми знайдемо в поточній директорії всі файли з ім'ям test * .py і запустимо всі тести. За замовчуванням, тести повідомлять що-небудь, тільки в разі невдачі.

Якщо ви хочете отримувати більше інформації про тести ви повинні змінити значення параметра verbosity. Наприклад, для виведення списку успішних і неуспішних тестів (і всю інформацію про те, як пройшла настройка бази даних) ви можете встановити значення verbosity рівним "2":

python manage.py test --verbosity 2

Доступними значеннями для verbosity є 0, 1 (значення за замовчуванням), 2 і 3.

Якщо ви хочете запустити підмножина тестів, тоді вам треба вказати повний шлях до вашого пакету, модулю / подмодулей, класу-нащадку TestCase, або методу:

```
# Run the specified module
python manage.py test catalog.tests

# Run the specified module
python manage.py test catalog.tests.test_models

# Run the specified class
python manage.py test catalog.tests.test_models.YourTestClass
```

Для перевірки поведінки відображення ми використовуємо тестовий кліет Django Client. Даний клас діє як спрощений веб-браузер який ми застосовуємо для імітації GET і POST запитів і перевірки відповідей. Про відповіді ми можемо дізнатися майже все, починаючи з низькорівневого HTTP (підсумкові заголовки і коди статусів) і аж до застосовуваних шаблонів, які використовуються для HTML-рендери, а також контексту, який передається в відповідний шаблон. Крім того, ми можемо відстежити послідовність перенаправлень (якщо є), перевірити URL-адреси та коди статусів на кожному кроці. Все це дозволить нам перевірити, що кожне відображення виконує те, що очікується.

10.3. Програма роботи

10.3.1. Ознайомитися з принципами конфігурування url-адрес веб-сайту.

10.3.2. Створити представлення для перегляду запису блогу та виведення списку усіх записів блогу, а також списку записів, згрупованих по категоріях.

10.3.3. Створити шаблони з використанням тегів та фільтрів для динамічних сторінок.

10.3.4. Ознайомитись із принципами наслідування та повторного використання коду.

10.3.5. Ознайомитися з принципами написання юніт-тестів.

10.4. Обладнання та програмне забезпечення

10.4.1. Персональний комп’ютер.

10.4.2. Інтерпритатор Python встановлений на ПК.

10.4.3. Середовище розробки програмного забезпечення PyCharm Community Edition (чи інше IDE, наприклад Eclipse, Sublime Text, Geany).

10.4.4. Web-фреймворк Django.

10.5. Порядок виконання роботи і опрацювання результатів

Тепер настав час оновити код в'юшкок, щоб «витягувати» список публікацій із таблиці в базі даних та відображати на сайті. Для початку додамо необхідні url-адреси:

```
# app_blog/urls.py
from django.urls import path

from .views import (HomePageView, ArticleDetail,
                    ArticleList, ArticleCategoryList)

urlpatterns = [
    path(r'', HomePageView.as_view()),
    path(r'articles', ArticleList.as_view(), name='articles-
list'),
    path(r'articles/category/<slug>',
         ArticleCategoryList.as_view(),
         name='articles-category-list'),
    path(r'articles/<year>/<month>/<day>/<slug>',
         ArticleDetail.as_view(),
         name='news-detail'),
]
```

Кожен із шаблонів має регулярний вираз, посилання на відповідну в'юшку та назву. Перша адреса була додана попередньо і відповідає головній сторінці вашого блогу. Друга адреса відповідає (на етапі розробки при роботі з локальним сервером) http://127.0.0.1:8000/articles, і за цією адресою буде розміщено список усіх публікацій.

Третя адреса мітить шаблон. Нова річ у даному коді - це регулярний вираз із змінними всередині:

r'articles/<year>/<month>/<day>/<slug>'

У ньому частина в трикутних дужках визначає динамічну складову URL адреси - унікальний ідентифікатор публікації, який складається із дати публіації та слагу (відповідно до того, що повертає функція *get_absolute_url* моделі *Article*). Адреса типу "arcticles/2020/01/01/testova-publikaciya/" задовольнить даний регулярний вираз. А рядок всередині кутових дужок передасть отримані змінні дати та слагу у функцію в'юшки.

Додавши дані рядки отримаємо помилку, оскільки в аплікейшині відсутні вьюшки ArticleDetail, ArticleList. Додамо

ïx:

```
# app_blog /views.py
from django.shortcuts import render
from django.views.generic import TemplateView, ListView,
DateDetailView

from .models import Article


class HomePageView(TemplateView):
    def get(self, request, **kwargs):
        return render(request, 'index.html', context=None)


class ArticleDetail(DateDetailView):
    model = Article
    template_name = 'article_detail.html'
    context_object_name = 'item'
    date_field = 'pub_date'
    query_pk_and_slug = True
    month_format = '%m'
    allow_future = True

    def get_context_data(self, *args, **kwargs):
        context = super(ArticleDetail,
self).get_context_data(*args, **kwargs)
        try:
            context['images'] = context['item'].images.all()
        except:
            pass

        return context


class ArticleList(ListView):
    model = Article
    template_name = 'articles_list.html'
    context_object_name = 'items'

    def get_context_data(self, *args, **kwargs):
    context = super(ArticleList,
self).get_context_data(*args, **kwargs)
    try:
```

```
        context['category'] =
Category.objects.get(slug=self.kwargs.get('slug'))
    except Exception:
        context['category'] = None

    return context


    def get_queryset(self, *args, **kwargs):
        articles = Article.objects.all()

        return articles


class ArticleCategoryList(ArticleList):

    def get_queryset(self, *args, **kwargs):

        articles = Article.objects.filter(

category__slug__in=[self.kwargs['slug']]).distinct()

        return articles
```

Додавання сторінок

Додамо сторінку на якій буде розміщено список всіх публікацій. У вашій папці templates додайте файл articles_list.html. У ньому вставте наступний HTML код:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Публікації</title>
</head>
<body>
   Публікації <br/>

  {% if category %} {{ category }} {% endif %}

  {% for item in items %}
      <div class="articles-row">
        <a href="{{ item.get_absolute_url }}">
```

```
            <h4>{{ item.title }}</h4>
          </a>
          <h5>
            {{ item.pub_date|date:"j E Y"|safe|linebreaks }}
          </h5>
          <p>
            {{ item.description|safe|escape|striptags|truncatewords_html:32 }}
          </p>
          <div class="container-image">
            <img class="img" src="{{ item.images.all.0.image.url }}" alt="" />
          </div>
          <div class='clearfix'></div>
      </div>
      {% endfor %}
  </body>
  </html>
```

А також сторінку для окремої публікації **article_detail.html**

```
<!DOCTYPE html>
<html lang="en">

<head>
   <meta charset="UTF-8">
   <title>{{ item.title }}</title>
</head>

<body>
   {% load static %}
   <div>
      <ol class="breadcrumb">
         <li><a href="/">Головна</a></li>
         <li><a href="{% url 'articles-list' %}">Публікації</a></li>
         <li><a href="{{ item.category.get_absolute_url }}">{{ item.category|upper
  }}</a></li>
         <li>{{ item.title|upper }}</li>
      </ol>
      <div>
         <h3>
            {{ item.title }}
         </h3>
         <h5>
```

```
          {{ item.pub_date|date:"d E Y"|safe|linebreaks }}
          </h5>
        </div>
        <div>
          {{ item.description|escape|safe }}

            {% if item.images.all %}
               {% include 'fotorama.html' with images=images %}
            {% endif %}

        </div>
        <div class='clearfix'></div>
    </div>
</body>
</html>
```

На цій сторінці є підключення іншої сторінки fotorama.html за умови що для публікації додані зображення (тег {% if item.images.all %}…{% endif %}). Вміст файлу fotorama.html:

```
{% spaceless %}
{% load static %}

{% block head %}
<!-- jQuery 1.8 or later, 33 KB -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js">
</script>
<!-- Fotorama from CDNJS, 19 KB -->
<link          href="https://cdnjs.cloudflare.com/ajax/libs/fotorama/4.6.4/fotorama.css"
  rel="stylesheet">
<script src="https://cdnjs.cloudflare.com/ajax/libs/fotorama/4.6.4/fotorama.js">
</script>
{% endblock %}

<div class="fotorama">
  {% for image in images %}
    <img src='{{ image.image.url }}'
        alt="{{ image.description }}"
        data-caption="{{ image.description }}">
  {% endfor %}
</div>
```

```
{% endspaceless %}
```

В даному випадку ми просто підключили готову галерею https://fotorama.io/, додавши відповідні посилання в секцію *head* та вказавши клас *class=”fotorama”* для блока (div), що містить усі зображення публікації.

Також модифікуємо головну сторінку. Для цього змінимо представлення HomePageView (app_blog/views.py) та index.html.

```
class HomePageView(ListView):
    model = Article
    template_name = ‘index.html’
    context_object_name = ‘categories’

    def get_context_data(self, **kwargs):
        context = super(HomePageView,
                self).get_context_data(**kwargs)
        context[‘articles’] =\
           Article.objects.filter(main_page=True)[:5]

        return context

    def get_queryset(self, *args, **kwargs):
        categories = Category.objects.all()

        return categories
     ….
```

index.html:

```
<!DOCTYPE html>
<html lang=”uk”>
<head>
    <title>Blog</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
</head>
  <body>
Категорії
  {% if categories %}
{% for item in categories %}
<div>
    <a href="{{ item.get_absolute_url }}">
     <h4>{{ item.category }}</h4>
```

```
    </a>
  </div>
  {% endfor %}
{% endif %}
{% if articles %}
{% for item in articles %}
<div>
     <a href="{{ item.get_absolute_url }}">
      <h4>{{ item.title }}</h4>
   </a>
     <h5>
       {{ item.pub_date|date:"j E Y"|safe|linebreaks }}
   </h5>
     <p>
       {{ item.description|safe|escape|striptags|truncatewords_html:32 }}
   </p>
  </div>
  {% endfor %}
{% endif %}
  <a href="{% url 'articles-list' %}">
    <h4>Всі публікації</h4>
  </a>

  </body>
  </html>
```

Для того, щоб екземпляри класу Category повертали url-адресу необхідно додати у відповідний клас метод get_absolute_url, який ми викликаємо в темплейті

<a href="{{ item.get_absolute_url }}">

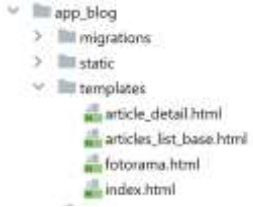
Також для необхідно додати менеджер моделі, інакше виклик Category.objects.all() не поверне список всіх категорій.

Тобто у файлі models.py оновіть клас Category наступним чином:

```
class Category(models.Model):
    category = models.CharField(u'Категорія',
       max_length=250, help_text=u'Максимум 250 символів')
    slug = models.SlugField(u'Слаг')
    objects = models.Manager()
```

```
    class Meta:
        verbose_name = u'Категорія для публікації'
        verbose_name_plural = u'Категорії для публікацій'

    def __str__(self):
        return self.category

    def get_absolute_url(self):
        try:
            url = reverse('articles-category-list',
                          kwargs={'slug': self.slug})
        except:
            url = "/"
        return url
```

Тепер у директорії tamplates повинні знаходитись 4 html-файли:



Запуск сервера та перехід на сторінку http://127.0.0.1:8000/articles буде відображати шаблон зі списком публікацій. На головній сторінці буде список всіх категорій, за яким можна переходити на відповідні сторінки зі списками публікацій, що відносяться до вказаної категорії. Також на головній сторінці розміщено 5 найновіших публікацій з позначкою *'показувати на головній сторінці'*.

Натиснувши на заголовок публікації, ви будете переправлені на сторінку конкретної публікації, яка міститиме навігаціну секцію, заголовок, дату та текст публікації, а також «карусель» із зображень, що додані до конкретної публікації.

На даному етапі зовнішній вигляд сторінок не оформлений. Клієнтську частину сайту розроблятимемо далі.

Завдання: Продумайте та доповніть навігацію по сайту, наприклад, можливість повернутись зі сторінки перегляду списку публікацій на головну.

Тестування сайту

На даний момент ми будемо працювати в файлі **tests.py** всередині застосунку **app_blog**:

```
from django.test import TestCase
from django.urls import reverse


class HomeTests(TestCase):
    def test_home_view_status_code(self):
        url = reverse('home')
        response = self.client.get(url)
        self.assertEquals(response.status_code, 200)
```

Це надзвичайно простий тестовий випадок (test case), але надзвичайно корисний. Ми перевіряємо *код статусу* відповіді. Код статусу 200 означає **успіх**.

Ми можемо перевірити код статусу відповіді в консолі, після запуску сервера командою runserver і переходу у браузері на головну сторінку http://localhost:8000/ в консолі побачите:

"GET / HTTP/1.1" **200** 1144

Якби було б неперехоплене виключення (uncaught exception) чи щось інше, Django замість цього повернув би код статусу **500**, що означає **внутрішню помилку сервера (Internal Server Error)**. Тепер уявіть, що наш застосунок має 100 представлень. Якби ми написали лише цей простий тест для всіх наших представлень, за допомогою лише однієї команди, ми могли б перевірити, чи всі представлення повертають код успіху. Без автоматизації тестів нам потрібно буде перевіряти кожну сторінку окремо.

Запускаємо тест командою (віртуальне оточення повинно бути активоване):

(myvenv) D:\work\myblog\mysite>**python manage.py test**

Якщо виконавши цю команду отримали помилку доступу до тестової бази даних, в даному випадку ‘test_mysite_db’:

Got an error creating the test database: (1044, "Access denied for user ‘mysite_usr’@’localhost’ to database ‘test_mysite_db’")

необхідно надати права вказаному в налаштуваннях (settings.py DATABASES, виконувались в лабораторній роботі 8). Для цього в каталозі C:\Program Files\MySQL\MySQL Server 8.0\bin виконуємо вхід до MySQL Server командою mysql -u root -p

C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u root -p
Enter password: *******

і надаємо права вказаному в налаштуваннях користувачу mysite_usr на всі таблиці бази даних test_mysite_db:

grant all privileges on test_mysite_db.* to ‘mysite_usr’@’localhost’ with grant option;

Запускаємо ще раз:
(myvenv) D:\work\myblog\mysite>python manage.py test

Даний тест повинен «впасти» з помилкою:

Traceback (most recent call last):
  File "D:\work\myblog\mysite\app_blog\tests.py", line 7, in test_home_view_status_code
    url = reverse(‘home’)
  File "D:\work\myblog\myvenv\lib\site-packages\django\urls\base.py", line 87, in reverse

```
        return iri_to_uri(resolver._reverse_with_prefix(view, prefix,
*args, **kwargs))
      File "D:\work\myblog\myvenv\lib\site-
packages\django\urls\resolvers.py", line 677, in _reverse_with_p
refix
        raise NoReverseMatch(msg)
    django.urls.exceptions.NoReverseMatch: Reverse for 'home' not
found. 'home' is not a valid view function or pattern name.

    ----------------------------------------------------------------------
    Ran 1 test in 0.026s

    FAILED (errors=1)
    Destroying test database for alias 'default'...
```

Це відбулось через те, що в аплікейшині немає url з name='home'. У файлі app_blog/urls.py змінимо рядок з першим url - path(r'', HomePageView.as_view()) наступним чином:

```
urlpatterns = [
    path(r'', HomePageView.as_view(), name='home'),
```

Ще раз запустіть тест і переконайткс що він пройшов успішно

```
    Creating test database for alias 'default'...
    System check identified no issues (0 silenced).
    .
    ----------------------------------------------------------------------
    Ran 1 test in 0.057s

    OK
    Destroying test database for alias 'default'...
```

Тепер ми можемо перевірити, чи повернув Django правильне представлення функції для запитуваної URL-адреси. Це також корисний тест, тому що, просуваючись з розробкою, ви побачите, що модуль **urls.py** може стати дуже великим і

складним. Конфігурація URL — це вирішення регулярних виразів. Є деякі випадки, коли ми вже маємо допустиму URL-адресу, тому Django може повернути неправильну функцію представлення. В клас HomeTests із tests.py додамо ще один тест:

```
from django.test import TestCase
from django.urls import reverse, resolve

from .views import HomePageView


class HomeTests(TestCase):
    def test_home_view_status_code(self):
        url = reverse('home')
        response = self.client.get(url)
        self.assertEquals(response.status_code, 200)

    def test_home_url_resolves_home_view(self):
        view = resolve('/')
        self.assertEquals(view.func.view_class,
HomePageView)
```

У другому тесті ми використовуємо функцію resolve. Django використовує її для перевірки відповідності запитуваної URL-адреси зі списком URL-адрес, перерахованих у модулі **urls.py**. Цей тест дозволить переконатися, що URL /, яка є кореневою URL-адресою, повертає представлення home.

Протестуємо знову:

```
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..
----------------------------------------------------------------------
Ran 2 tests in 0.024s

OK
Destroying test database for alias 'default'...
```

Щоб побачити більш детальну інформацію про виконання тесту, встановіть **verbosity** на більш високий рівень:

(myvenv) D:\work\myblog\mysite>**python manage.py test --verbosity=2**

Для того щоб написати тест для url з параметрами, необхідно ці параметри спеціальним чином передати. Наприклад додамо тест для шаблону

```
path(r'articles/category/<slug>', 
     ArticleCategoryList.as_view(), 
     name='articles-category-list'),
```

який в якості параметра приймає рядковий парамер. Якщо спробувати написати тест, аналогічно до `test_home_view_status_code`:

```
def test_category_view_status_code(self):
    url = reverse('articles-category-list')
    response = self.client.get(url)
    self.assertEquals(response.status_code, 200)
```

І запустити тести, отримаємо один фейл (fail, тобто тест не пройшов) з повідомленням, який саме тест не пройшов і причиною. Виправимо його, додавши аргумент для шаблону (зверніть увагу ми передаємо в якості `args` кортеж, тому в дужках стоїть кома після аргумента – так в Python створюється кортеж з одного елемента):

```
def test_category_view_status_code(self):
    url = reverse('articles-category-list', args=('name',))
    response = self.client.get(url)
    self.assertEquals(response.status_code, 200)
```

Завдання: додайте юніт тести для всіх url. Перейменуйте tests.py tests_urls.py (префікс tests_ на початку обов'язковий) і переконайтесь що тести так само виконуються, запустивши їх командою python manage.py test

Створіть ще один файл з тестами в директорії app_blog для тестування моделей з назвою tests_model.py. Наприклад, щоб протесувати функцію get_absolute_url моделі Category напишемо наступний тест:

```
from django.test import TestCase

# Create your tests here.

from .models import Category

class CategoryModelTest(TestCase):

    @classmethod
    def setUpTestData(cls):
        #Set up non-modified objects used by all test
        Category.objects.create(category='Innovations',
slug='innovations')

    def test_get_absolute_url(self):
        category=Category.objects.get(id=1).

        self.assertEquals(category.get_absolute_url(),
        '/articles/category/innovations')
```

В даному випадку до виконання тесту (setUpTestData) створюється запис для категорії "Innovations" в тестовій базі даних. Далі в тесті (test_get_absolute_url) перевіряється значення, яке повертає метод get_absolute_url для інстанса класу Category. Закомітьте зміни в репозиторій на GitHub та додайте посилання на репозиторій до звіту.

10.6. Контрольні запитання.

10.6.1. В якому файлі зберігаються представлення аплікейшина?

10.6.2. В якому модулі мститься представлення ListView.

10.6.3. Що таке мова шаблонів Django?

10.6.4. Для чого використовується тег truncatewords_html?

10.6.5. Для чого проводиться тестування сайту?
10.6.6. Що таке юніт-тест?
10.6.7. Якою командою запускаються тести?

Література:

1. Лутц М. Программирование на Python, том I, 4-е издание.– Пер. сангл. – СПб.: Символ-Плюс, 2011.– 992с.
2. Документація Django [Електронний ресурс] / Режим доступу : https://djbook.ru/rel3.0

Лабораторна робота №11
Розробка клієнтської частини веб-застосування. Робота зі статичними файлами

11.1. Мета роботи

Познайомитись з базовими принципами оформлення веб-сторінок та розробити власний графічний дизайн для персонального блогу.

11.2. Теоретичні відомості

Веб-додатки зазвичай вимагають різні додаткові файли для своєї роботи (зображення, CSS, Javascript і ін.). В Django їх прийнято називати "статичними файлами" (або "статика").

Статичні файли — це CSS, JavaScript, шрифти, зображення або будь-які інші ресурси, які ми можемо використовувати для створення користувацького інтерфейсу.

Django не обслуговує ці файли. За винятком процесу розробки, щоб полегшити нам життя. Але Django надає деякі функції, які допомагають нам керувати статичними файлами. Ці функції доступні в застосунку **django.contrib.staticfiles**, вже вказаного в конфігурації INSTALLED_APPS.

Bootstrap

Bootstrap - це фреймворк, набір HTML + CSS інструментів і шаблонів для верстки і більш ефективного і швидкого створення сайтів і веб-додатків більш ефективно і швидко доступний для використання за відкритою ліцензією.

Bootstrap - інтуїтивно простий і потужний інтерфейсний фрейморк, що підвищує швидкість і полегшує розробку веб-додатків для всіх пристроїв.

Bootstrap легко і ефективно масштабує проект з однією базою коду, від телефонів і планшетів до настільних комп’ютерів. Цей фреймворк дуже динамічний і регулярно оновлюваний, тому не всі його функції можуть коректно підтримуватися старими браузерами.

Адаптивне верстання – підхід, що припускає зміну дизайну залежно від поведінки користувача, розміру екрана, платформи і орієнтації девайса. Іншими словами, сторінка повинна автоматично підлаштовуватися під дозвіл, змінювати розмір картинок і так далі. Це дозволить усунути потребу в розробці дизайну для кожного нового пристрою, що з'являється у продажу.

Регулювання дозволу екрана

За допомогою CSS медіа-запитів веб-дизайнери можуть створювати стилі для певних пристроїв, а також при виконанні певних умов (наприклад, при певній ширині, висоті або орієнтації екрану пристрою).

Найбільш часто використовувані медіа-запити в веб-дизайні.

```
/* ----- Настільні комп'ютери та ноутбуки ----------- */
@media only screen
and (min-width: 1224px) {
  /* ваші CSS-стилі тут */
}

/* ----- Девайси з великими екранами ----------- */
@media only screen
and (min-width: 1824px) {
  /* ваші CSS-стилі тут */
}

/* ----- Смартфони  ----------- */
@media only screen
and (min-device-width: 320px)
and (max-device-width: 480px) {
  /* ваші CSS-стилі тут */
}
```

CSS медіа-запити підтримуються в Chrome 1+, Firefox 3.6+, Internet Explorer (IE) з версії 9+, Opera 10+, Safari 4+, а також в смартфонах та інших мобільних пристроях.

11.3. Програма роботи

11.3.1. Ознайомитися з принципами роботи з CSS/HTML-фреймворком Bootstrap.

11.3.2. Розробити макет та виконати кодування сторінок веб-сайту з використанням використовувати засоби Twitter Bootstrap.

11.3.3. Завершити інформаційне (графічне та текстове) наповнення сторінок Web-сайту.

11.4. Обладнання та програмне забезпечення

11.4.1. Персональний комп'ютер.

11.4.2. Інтерпритатор Python встановлений на ПК.

11.4.3. Середовище розробки програмного забезпечення PyCharm Community Edition (чи інше IDE, наприклад Eclipse, Sublime Text, Geany).

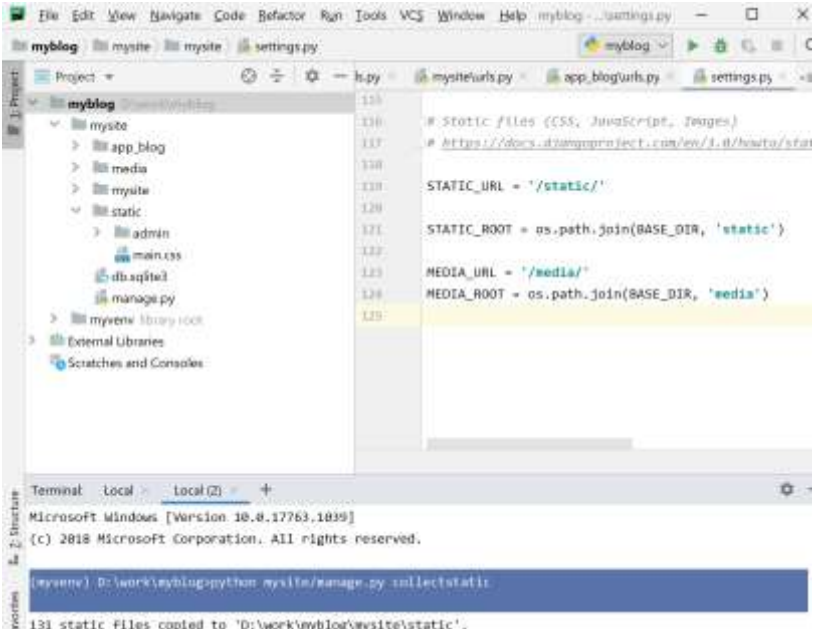
11.4.4. Web-фреймворк Bootstrap.

11.5. Порядок виконання роботи і опрацювання результатів

Встановлення статичних файлів

За наявності такої кількості бібліотек фронтенд компонентів, немає ніякої необхідності того, щоб продовжувати рендерити основні HTML-документи. Ми з легкістю можемо додати Bootstrap 4 до нашого проекту. Bootstrap — це інструментарій з відкритим вихідним кодом для розробки з HTML, CSS та JavaScript.

В терміалі, знаходячись в кореневому каталозі (де розміщений файл manage.py) проекту myblog виконайте команду ***python manage.py collectstatic*** (з активованим віртуальним оточенням) :



Зверніть увагу, що в кореневому каталозі з'явилась папка static з директорією admin. Так само додамо директорію static в app_blog.

Тут будемо зберігати статичні ресурси аплікації. Створимо в папці static два каталог css і js, щоб окремо зберігати файли стилів та скрипти. Порожній поки файл main.css перемістіть в теку css.

Перейдіть на getbootstrap.com завантажте останню версію:

Завантажте скомпільовані версії CSS та JS. На комп'ютері вилучіть файл **bootstrap-4.4.1-beta-dist.zip**, завантажений з сайту Bootstrap, скопіюйте файл **css/bootstrap.min.css** у теку static/css проекту. Також потрібно скачати останню версію jQuery в теку app_blog/static/js під назвою jquery.js

Тепер нам потрібно завантажити статичні файли (файл CSS Bootstrap) в наш шаблон. При цьому створимо спільний шаблон, де будемо підключати статику і який міститиме загальні для усіх сторінок сайту елементи base.html та підключимо його у всіх інших сторінках айту, замість того щоб на кожній сторінці дублювати підключення стилів та скриптів:

templates/home.html

```
<!DOCTYPE html>
<html lang="uk">
{% load static %}
<head>
      <title>Blog</title>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
   <!-- Include Styles -->
   <!DOCTYPE html>
   <link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
   <link type="text/css"  href="{% static 'css/main.css' %}" rel="stylesheet">
</head>
<body>
<div class="container">
   <h1> Персональний блог</h1>
   {% block content %}
   {% endblock %}
</div>
<!-- Javascripts Section -->
  <script src="{% static 'js/jquery.js' %}"
 crossorigin="anonymous"></script>
<script src="{% static 'js/bootstrap.min.js' %}" crossorigin="anonymous"></script>
</body>
</html>
```

Спочатку ми завантажуємо теги шаблону застосунку Static Files, використовуючи {% load static%} на початку шаблону.

Тег шаблону {% load static %} використовується для складання URL, де живе ресурс. У цьому випадку {% static ‘css/bootstrap.min.css’%} поверне app_blog/static/css/bootstrap.min.css, що еквівалентно http://127.0.0.1:8000/app_blog/static/css/bootstrap.min.css.

Тег шаблону {% static %} використовує конфігурацію STATIC_URL у settings.py, щоб скласти кінцеву URL-адресу. Наприклад, якщо б ви розмістили свої статичні файли в субдомені, як https://static.example.com/, то ми встановили б STATIC_URL = https: //static.example.com/, а потім {% static ‘css/bootstrap.min.css’ %} повернув би https://static.example.com/app_blog/static/css/bootstrap.min.css.

Теги {% block content %} {% endblock %} вказують, куди буде підставлятись контент сторінки, яка цей шаблон викликає за допомогою тега {% extends “base.html” %}.

Відповідно index.html потрібно відредагувати наступним чином

```
{% extends "base.html" %}
{% load static %}
{% block content %}
{% spaceless %}

<h2>Категорії</h2>
{% if categories %}
{% for item in categories %}
<div>
   <a href="{{ item.get_absolute_url }}">
      <h4>{{ item.category }}</h4>
   </a>
</div>
{% endfor %}
{% endif %}
{% if articles %}
{% for item in articles %}
<div class="article-block">
   <a href="{{ item.get_absolute_url }}">
      <h4>{{ item.title }}</h4>
   </a>
   <h5>
```

```
        {{ item.pub_date|date:"j E Y"|safe|linebreaks }}
    </h5>
    <p>
        {{ item.description|safe|escape|striptags|truncatewords_html:32 }}
    </p>
</div>
{% endfor %}
{% endif %}
    <a href="{% url 'articles-list' %}">
        <h4>Всі публікації</h4>
    </a>
{% endspaceless %}
{% endblock %}
```

Те саме для сторінок article_ist

```
{% extends "base.html" %}
{% load static %}
{% block content %}
{% spaceless %}

<h1>Публікації</h1>
<br/>
{% if category %} {{ category }} {% endif %}

{% for item in items %}
<div class="articles-row">
    <a href="{{ item.get_absolute_url }}">
        <h4>{{ item.title }}</h4>
    </a>
    <h5>
        {{ item.pub_date|date:"j E Y"|safe|linebreaks }}
    </h5>
    <p>
        {{ item.description|safe|escape|striptags|truncatewords_html:32 }}
    </p>
    <div class="container-image">
        <img class="img" src="{{ item.images.all.0.image.url }}" alt=""/>
    </div>
    <div class='clearfix'></div>
</div>
{% endfor %}
```

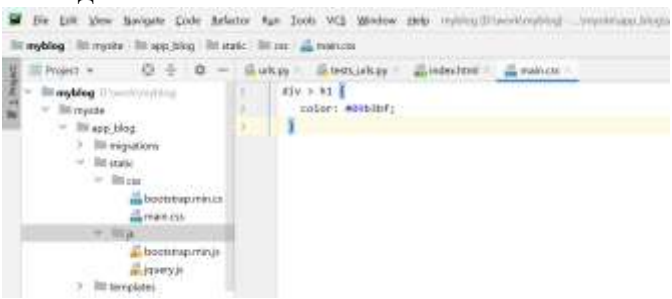
```
{% endspaceless %}
{% endblock %}
```

та article_detail.html

```
{% extends "base.html" %}

{% load static %}

{% block content %}
{% spaceless %}

<div>
   <ol class="breadcrumb">
      <li><a href="/">Головна</a></li>
      <li><a href="{% url 'articles-list' %}">Публікації</a></li>
      <li>{{ item.title|upper }}</li>
   </ol>
   <div>
      <h3>
         {{ item.title }}
      </h3>
      <h5>
         {{ item.pub_date|date:"d E Y"|safe|linebreaks }}
      </h5>
   </div>
   <div>
      {{ item.description|escape|safe }}

      {% if item.images.all %}
      {% include 'fotorama.html' with images=images %}
      {% endif %}

   </div>
   <div class='clearfix'></div>
</div>
{% endspaceless %}
{% endblock %}
```

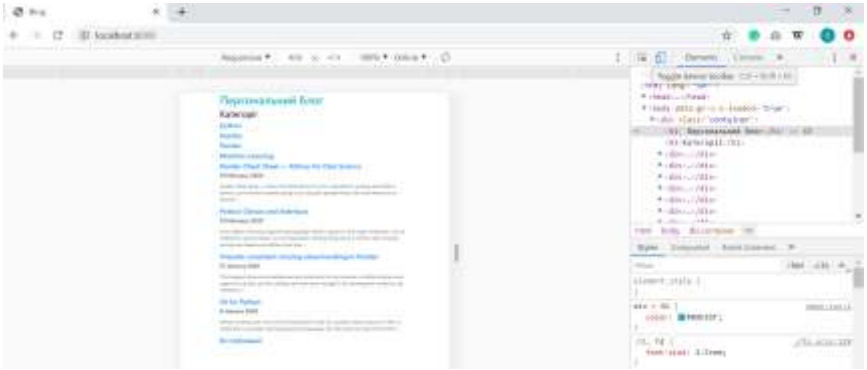
Оновлюючи сторінку 127.0.0.1:8000, можна побачити, що все працює. Тепер кожна зі сторінок сайту має напис «Персональний блог».

Додамо у файл main.css який-небуть стиль для перевірки коректного підключення:



В даному випадку ми встановили колір для всіх заголовків h1. Зверніть увагу на розміщення файлів в проекті. Перезапустіть сервер та перейдіть на головну сторінку вашого блогу.

Якщо попередні кроки були зроблені без помилок, ви побачите, що сторінки блогу змінились. Ддо деяких елементів застосовані bootstrap стилі (наприклад, стандартий bootstrap class="breadcrumb" на сторінці перегляду конкретної публікації article_detail.html). Загаловок рівня h1 змінив колір відповідно до того, як було задано у main.css. Зверніть увагу, що для верстки і перевірки стилів у браузері Chrome зручно використовувати *Консоль розробника* (F12). Наприклад, натиснувши  можна переглянути, як виглядатиме ваш блог на різних екранах (планшет, телефон):



Тепер ми можемо відредагувати шаблон таким чином, щоб скористатися перевагами Bootstrap CSS.

Додамо на головну сторінку навігаційну панель змість простого списку категорій:

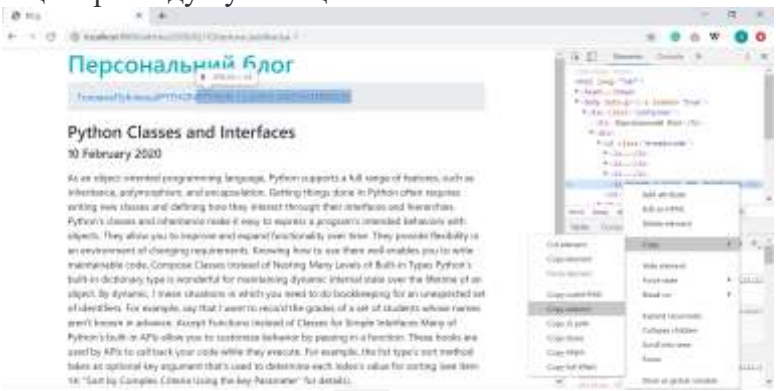
У файлі index.html замінимо рядки:

```
<h2>Категорії</h2>
{% if categories %}
  {% for item in categories %}
        <div>
          <a href="{{ item.get_absolute_url }}">
             <h4>{{ item.category }}</h4>
          </a>
        </div>
  {% endfor %}
{% endif %}
```

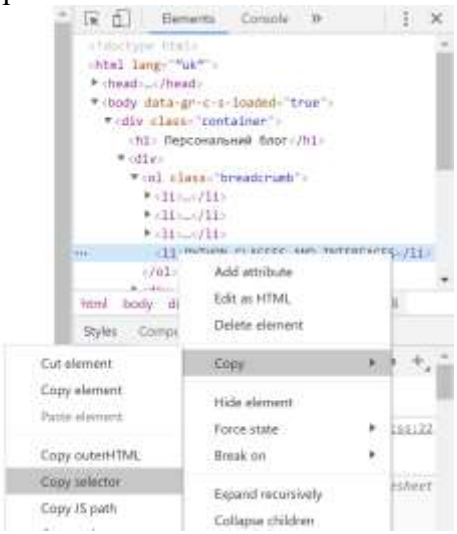
на наступні:

```
{% if categories %}
<nav class="navbar navbar-default" role="navigation">
  <div class="container">
    {% for item in categories %}
    <a href="{{ item.get_absolute_url }}">
       <h4>{{ item.category }}</h4>
    </a>
{% endfor %}
  </div>
</nav>
{% endif %}
```

Зробимо відступи між назвами в навігаційному елементі на сторінці перегляду публікації:



Використавши курсор  , знаходимо цей елемент у вікні навігації по сторінці, та викликавши контекстне меню, копіюємо необхідний селектор



Тепер цей селектор вставляємо у файл зі стилями main.css та надаємо необхідні відступи:

main.css:

```
div > h1 {
   color: #09b3bf;
}
```

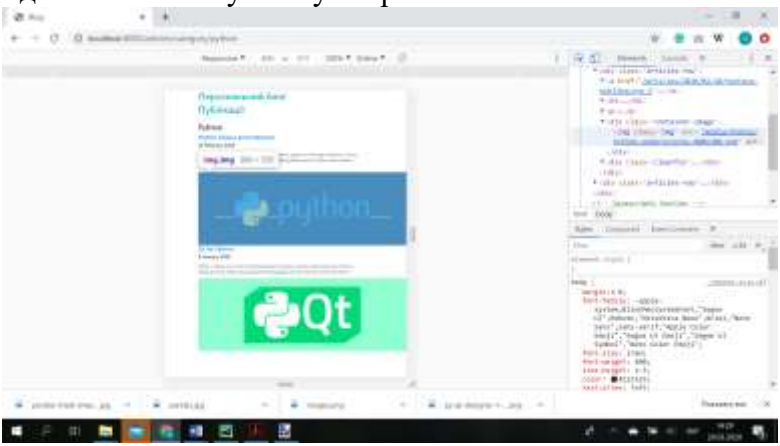
```
body > div > div > ol > li {
    margin: 0 5px;
}
```

Зверніть увагу, що таке копіювання стилів також влючає повне визначення певного елементу в тому числі з псевдоселектором:

```
body > div > div > ol > li:nth-child(1)
```

Псевдоселектор :nth-child(1) означає певний визначений елемент списку, щоб застосувати стиль (відступ в даному випадку) до всіх елементів списку, псевдоселектор необхідно видалити.

Виправимо помилки у верстці. Зверніть увагу як розміщені зображення на сторінці перегляду списку публікацій. Вони вставлені в шаблон за допомогою тега <img> в оригінальному розмірі, відповідно якщо зображення перевищує розміром розмір екрану (особливо це стосується мобільної версії), буде виглядати як на наступному зображенні



Розмір можна коригувати за допомогою атрибутів width і height, проте такий підхід викличе необхідність кожного разу повертатись до html файлів при потрбі редагувати стилі. Правильно всі стилі прописувати окремо у файлі чи файлах стилів (в даному випадку main.css) і виставляти розміри не у пікселях, а у відсотках, щоб отримати гнучкість у шаблоні.

Додайте у файл зі стилями наступні рядки:

```
div.container-image > img {
    width: 100%;
}
```

Перевантаживши сторінку сторінку, переконайтесь, що тепер зображення масштабуються відповідно до розміру екрана.

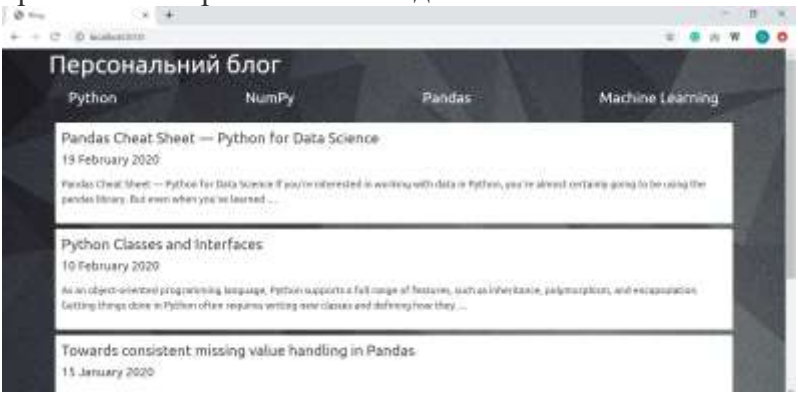
За домомогою власних стилів можна задати фон для певного елемента. Наприклад додамо фон для всього сайту. Зображення необхідно помістити в папку зі статикою (створимо окремий каталог для того щоб зберігати статичні зображення, адже їх кількість може в процесі розробки в залежності від макета зрости).

У файлі main.css додамо властивість background-image для body та ще декілька властивостей, наприклад імпорт та встановлення шрифтів, тіні для блоків та інше:

```
@import
url('https://fonts.googleapis.com/css?family=Ubuntu');

@media (max-width: 480px) {
  h1 {
    font-size: 18px;
  }
}
body {
    background-image: url('../img/bg.jpg');
    background-repeat: repeat-y;
}
div > h1 {
```

```
    color: #fff;
    font-family: ‘Ubuntu’, sans-serif;
}
body > div > nav > div > a > h4 {
    color: #fff;
    font-family: ‘Ubuntu’, sans-serif;
}
.article-block {
    background: #fff;
    color:  #383838;
    font-family: ‘Ubuntu’, sans-serif;
    box-shadow: 2px 2px 1px #ccc;
    padding: 10px;
    margin: 10px;
}
.article-block a {
    color: #383838;
    font-family: ‘Ubuntu’, sans-serif;
}
body > div > div > ol > li {
     margin: 0 5px;
}
div.container-image > img {
     width: 100%;
}
```

Тепер головна сторінка має вигляд:



Завдання: розробити клієнтську частину персонального блогу, додавши фон, рисунки, відступи, вирівнювання і т.і. Завершити наповнення інформацією через адмін-інтерфейс.

11.6. Контрольні запитання

11.6.1. Що таке статичні файли.
11.6.2. Що таке Bootstrap.
11.6.3. Для чого використовується тег {% extends … %}.
11.6.4. Для чого використовується тег {% load static %}.

Література:

1. https://www.w3schools.com/
2. https://bootstrap-4.ru/docs/4.4/getting-started/introduction/